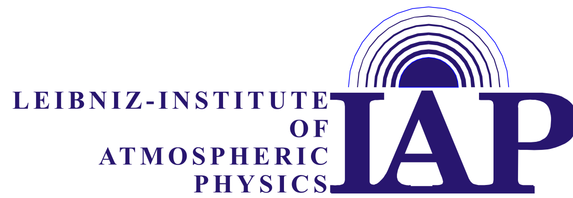

Master-Thesis

Animation von multidimensionalen atmosphärischen Parametern zur Untersuchung von leuchtenden Nachtwolken

Dirk Rachholz
geb. am 28.09.1979 in Kühlungsborn

Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Fakultät für Ingenieurwissenschaften, Bereich EuI

Betreuer, Einrichtung: Prof. Dr. S. Pawletta, HS Wismar
Dr. U. Berger, Leibniz-Institut für Atmosphärenphysik e.V., Kühlungsborn
Dr. G. Baumgarten, Leibniz-Institut für Atmosphärenphysik e.V., Kühlungsborn

Kühlungsborn, 14. Januar 2010

Aufgabenstellung

Animation von multidimensionalen atmosphärischen Parametern zur Untersuchung von leuchtenden Nachtwolken.

Animation of multidimensional atmospherical parameters for investigation of Noctilucent clouds.

Schwerpunkte

- **Untersuchung der Rechnerstruktur**
  Die Datenarchivierung- und Netzwerkstruktur des IAP ist im Vorfeld zu untersuchen. Weiter soll untersucht werden wie die dem IAP zur Verfügung stehenden Großrechner optimal genutzt werden können. Die resultierenden Ergebnisse sollen zu einem Skript zusammengefasst werden, das IAP-Mitarbeitern als allgemeine Informationsquelle dient.
- **Animation von multidimensionalen atmosphärischen Parametern**
  Es soll eine Animation aus Datenbanken erstellt werden, die global, dreidimensional und zeitabhängig definierte meteorologische Daten enthält. Angestrebt wird eine Version, die ähnlich zum Strömungsfilm des ARD Wetterberichts aufgebaut ist.

Tag der Ausgabe: 15.07.2009
Tag der Abgabe : 15.01.2010

Prof. Dr.-Ing. habil. Lochmann
Vorsitzender des
Prüfungsausschusses

Prof. Dr.-Ing. S.Pawletta
Betreuer

Autorenreferat

In der Abteilung Optische Sondierungen des Instituts für Atmosphärenphysik in Kühlungsborn (IAP), wird für die bessere Erforschung der Prozesse in der Atmosphäre seit 2005 das atmosphärische Leibniz Institute Middle Atmosphere Modell (LIMA) genutzt. In der vorliegenden Master-Thesis wurden Methoden zur Erstellung einer atmosphärischen Visualisierung, die LIMA- Wind- und Temperaturdaten kombiniert, untersucht und entwickelt. Die Untersuchungen und die Implementierung sind dabei auf die zur Verfügung stehende Programmiersprache IDL ausgerichtet. Weiterhin wurde die dem IAP zur Verfügung stehende Netzwerkstruktur und Großrechentechnik untersucht und in einem Benutzerhandbuch beschrieben.

Abstract

The department Optical Soundings of the Institute of Atmospheric Physics of Kühlungsborn (IAP) uses an atmospheric circulation model called LIMA (Leibniz Institute Middle Atmosphere Model) in order to investigate the main physical processes of the upper atmosphere at high latitudes. In the present Master-Thesis software solutions (programming language IDL) have been developed to visualize combined wind and temperature data from the LIMA-data archive. The general network structure and mainframes of the IAP computing system have been discussed, the results have been described in detail in a user manual.

Inhaltsverzeichnis

1. Einleitung

In der Forschung werden täglich riesige Datenmengen durch verschiedene Messsysteme oder Modellrechnungen mittels leistungsfähiger Computer produziert, die in unverarbeiteter Form nur schlecht oder gar nicht interpretierbar sind. Um die Daten untersuchen zu können und die in ihnen enthaltenen Informationen erfassbar zu machen, sind effektive Analysemethoden unverzichtbar geworden.

Die Visualisierung von Daten, bezeichnet die bildliche Veranschaulichung ihrer relevanten Aspekte. Sie spielt heute in vielen Lebensbereichen eine wichtige Rolle da sie die Kommunikation und die Erkenntnis entscheidend erleichtert. Die Visualisierung gewinnt durch den ständig wachsenden technologischen Fortschritt bei der Erzeugung von Daten sowie auch bei der Generierung von Grafiken zunehmend an Bedeutung. Am Institut für Atmosphärenphysik in Kühlungsborn (IAP) werden mit einer Vielzahl von Messsystemen große Mengen an Daten erzeugt. Außerdem wird zusätzlich anhand von komplexen Computermodellen Forschung betrieben, die wiederum große Datenmengen generieren, sodass die Wissenschaftler auf spezielle Visualisierungssysteme angewiesen sind, die den Anforderungen einzelner Problemstellungen entsprechen. In dieses Gebiet fällt die Aufgabenstellung der vorliegenden Master Thesis. Es sollen für vorhandene numerische Simulationsergebnisse die Möglichkeiten der Visualisierung von atmosphärischen Strömungen in Form von Filmen untersucht und angewendet werden. Die Arbeit der Mitarbeiter des IAP soll zusätzlich unterstützt werden, in dem eine ausführliche Beschreibung der Netzwerk- und Großrechnerumgebung, mit praktischen Anwendungsbeispielen erstellt wird. Sie soll den Mitarbeitern des IAP zukünftig als Einführungsmaterial bzw. Nachschlagewerk zur Verfügung steht. Die Beschreibung der rechentechnischen Arbeitsumgebung des IAP ist aus administrativen und sicherheitstechnischen Gründen in leicht abgewandelter Form in den beiden ersten Kapiteln enthalten. Die vorliegende Arbeit baut zum Teil auf Erfahrungen und Ergebnissen vorausgegangener Arbeiten ([1],[2]) auf,

die am IAP Kühlungsborn angefertigt wurden.

1.1. Das Institut für Atmosphärenphysik

Die Arbeitsgebiete des IAP wurden in [2] bereits beschrieben und werden folgend aktualisiert wiedergegeben. Das IAP arbeitet auf dem Gebiet der Atmosphärenphysik, wobei der Schwerpunkt bei der Erforschung der Atmosphäre zwischen 10 und 100 km liegt [3]. Hierbei werden die Mesosphäre und die dynamischen Wechselwirkungen zwischen den verschiedenen Schichten der Atmosphäre besonders berücksichtigt. Ferner wird untersucht, ob es in der oberen Atmosphäre zu langfristigen Veränderungen kommt und ob diese u. U. zur frühzeitigen Warnung von Klimaänderungen genutzt werden können. Am IAP werden drei Schwerpunkte bearbeitet:

- Erforschung der Mesosphäre:
  Die Mesosphäre wird in verschiedenen geographischen Breiten experimentell mit Hilfe von Lidars, Radars und Höhenforschungsraketen untersucht, wobei der Schwerpunkt auf der thermischen und dynamischen Struktur der Mesopausenregion liegt. Darüber hinaus werden Modellrechnungen unterschiedlicher Komplexität zum tieferen Verständnis der Phänomene von NLC, PMSE und PMWE durchgeführt.

- Kopplung der atmosphärischen Schichten:
  Das Forschungsgebiet der Wechselwirkung von Troposphäre, Stratosphäre und Mesosphäre dient einem verbesserten Verständnis der Atmosphäre. Atmosphärische Wellen findet man auf sehr unterschiedlichen räumlichen und zeitlichen Skalen. Sie sind das zentrale Element der dynamischen Kopplung der einzelnen Schichten. Um Anregungsprozesse von Wellen im Einzelnen zu verstehen, werden Ergebnisse von Messungen und von zeitlich und räumlich hoch aufgelösten Modellen kombiniert.

- Trends in der mittleren Atmosphäre:
  Die Untersuchungen langfristiger Änderungen der Atmosphäre erfolgen sowohl aus grundlagenwissenschaftlichem als auch aus umweltpolitischem Interesse.

> Dazu werden die am IAP durchgeführten langzeitigen Beobachtungsreihen sowie Temperaturmessungen in der polaren Mesosphäre, die nun schon seit 1994 vorgenommen werden, im Hinblick auf Trends in der oberen Atmosphäre herangezogen.

Das IAP in Kühlungsborn ist intern in fünf Abteilungen gegliedert. Es unterteilt sich in eine Verwaltungsabteilung, eine Infrastrukturabteilung und drei Wissenschaftliche Abteilungen. Die Infrastruktur beinhaltet das Rechenzentrum, die Bibliothek und die Werkstatt. Der wissenschaftliche Bereich ist in die Abteilungen Optische Sondierungen, Radarsondierungen und Höhenforschungsraketen und Theorie und Modellierung gegliedert. Das IAP besitzt eine Außenstelle in Juliusruh auf Rügen, wo vor allem Radarmessungen vorgenommen werden. Eine weitere Außenstelle befindet sich bei Andenes, in Norwegen (s. Abb. 1.1). Dort werden die atmosphärenphysikalischen Prozesse anhand von Radars, optischen Messystemen sog. Lidars im ALOMAR Observatorium und Höhenforschungsraketen studiert. Zusätzlich besitzt das IAP mehrere hochauflösende Kammeras zur Beobachtung von leuchtenden Nachtwolken an verschiedennen Obsevatorien in Europa, mit denen NLC- Ereignisse dokumentiert werden können.

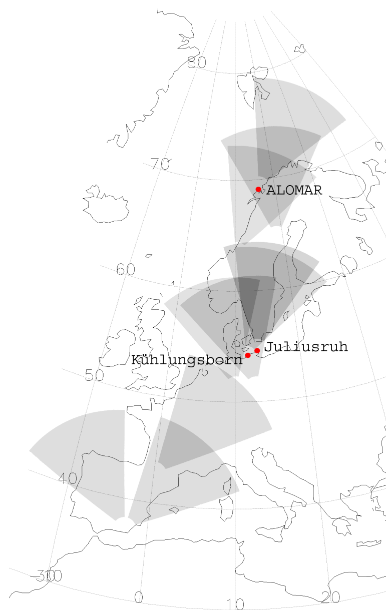

Abbildung 1.1.: Dargestellt ist die geografische Verteilung der einzelnen Außenstationen des IAP und die Kameras ,bzw. deren Sichtfelder in grau [4].

2. Datenarchivierungs- und Netzwerkstruktur am IAP

Die Arbeit mit Computern ist ein fester Bestandteil in der Wissenschaft geworden. Dies trifft insbesondere auch für die wissenschaftlichen Arbeiten am IAP zu. Dazu zählen rechenintensive Modellanimationen, deren Ein- und Ausgangsdaten ein immenses Datenvolumen darstellen, wie auch Datenerhebungen und Auswertungen von Messreihen, die in ihrer Effizienz und Genauigkeit, wie sie heutzutage vollzogen werden können, ohne Computer nicht möglich wären. Im folgenden Kapitel wird die Datenarchivierungs- und Netzwerkstruktur am IAP so beschrieben, dass neue Mitarbeiter einen möglichst einfachen Einstieg in ihre neue Arbeitsumgebung haben.

2.1. Netzwerkstruktur

Das IAP-Netzwerk umfasst die Rechnerumgebung im Institut in Kühlungsborn, in der Außenstelle Juliusruh auf Rügen, der Außenstelle in Andenes in Norwegen sowie einige Netzwerkkameras, die an verschiedenen Stellen in Europa verteilt sind. Die folgende Beschreibung bezieht sich allein auf die Rechnerumgebung des Institutes in Kühlungsborn.

Das gesamte Netzwerk (s. Abb. 2.1) verfügt über eine Bandbreite von 1 Gb/s und ist mit Switchen segmentiert. Es ist auf zwei Gebäude verteilt, zum Einen auf das Hauptgebäude mit den Büros und Laboren der Wissenschaftler sowie der Verwaltung. Zum Anderen auf das Rechenzentrum, das sich in unmittelbarer Nähe befindet, in dem die Großrechentechnik konzentriert ist. Die Netzwerke der beiden Gebäude sind mit 6×4 Gb/s Glasfaserleitungen verbunden. Im Hauptgebäude ist das Netz weiter

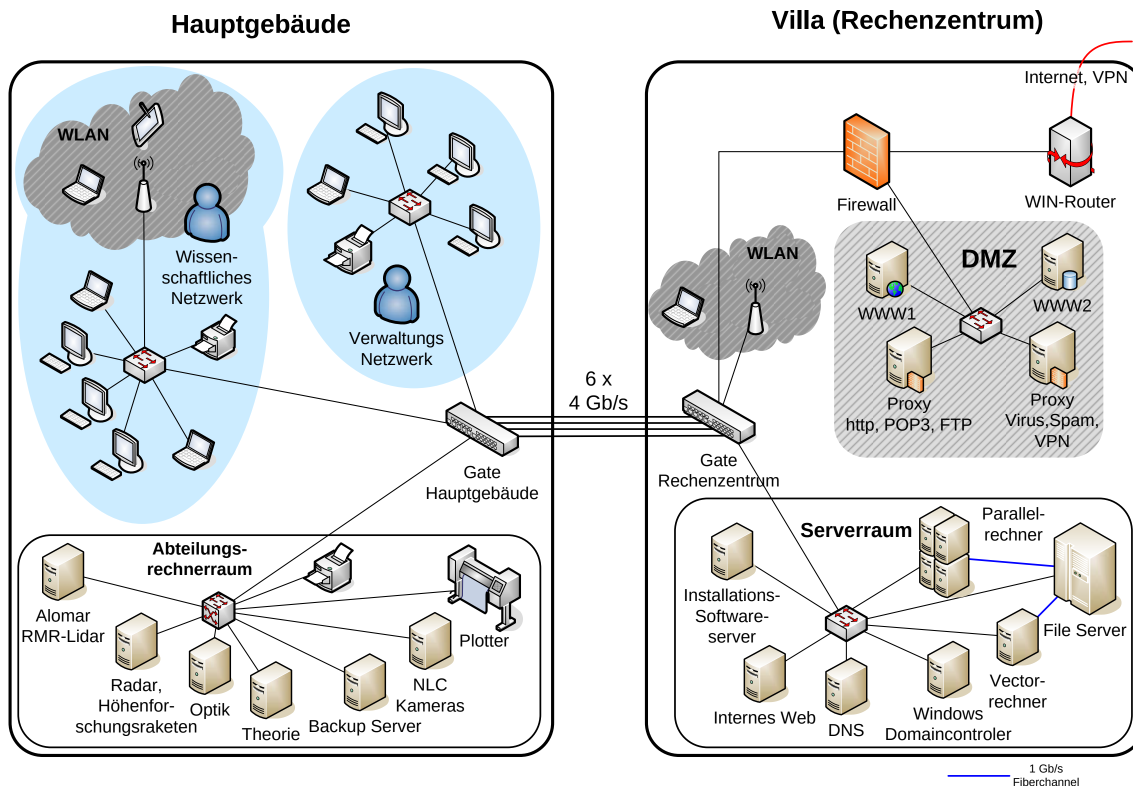

Abbildung 2.1.: Schematische Darstellung des IAP Netzwerkes dem ein Client- Server- Modell mit zentralem Fileserver zugrunde liegt. Aus administrativen Gründen ist es hier sehr allgemein dargestellt.

unterteilt in das wissenschaftliche und das Verwaltungsnetzwerk. Das wissenschaftliche Netzwerk verfügt über Local Area Network (LAN) -Anschlüsse mit festen und dynamisch vergebenen IP-Adressen, sowie einem Wireless-LAN (WLAN) mit mehreren Accesspoints. Es umfasst die Arbeitsplatzrechner in den Büros, diverse Steuerrechner in den Laboren, die Abteilungsrechner und eine größere Anzahl von Netzwerkdruckern. Die Arbeitsplatzrechner sind für gewöhnlich einfache PC's oder Notebooks, die für Microsoft Windows Desktopanwendungen zur Verfügung stehen bzw. mit denen per Remote-Login auf den Linux Abteilungsrechnern oder anderen Parallelrechnern im Rechenzentrum gearbeitet werden kann. Im Rechenzentrum sind Netzwerk-Management-Server, Parallel- und Vektorrechner als auch der zentrale Fileserver des IAP aus wartungstechnischen und administrativen Gründen platziert (s. Abb. 2.1).

Das Netz ist hier logisch in ein LAN, ein WLAN und eine DMZ (demilitarisierte Zone) aufgeteilt. Im Rechenzentrum-LAN sind Server und Rechner integriert, die für das Arbeiten innerhalb des IAP zur Verfügung stehen, bzw. für das Management des IAP-internen Netzes verantwortlich sind. Die DMZ beherbergt alle Management-Server, die möglichst auf sicherer Basis Dienste sowohl dem internen als auch dem externen Netz (Internet) zur Verfügung stellen. Ein Beispiel dafür ist der Mailserver, der ein Teil der DMZ ist. Er muss von außen erreichbar sein, da ansonsten E-Mails nicht zugestellt werden könnten. Anderseits müssen die Clients, die am LAN angeschlossen sind, ihre E-Mails abholen. Das interne Netz, die DMZ und das Internet sind jeweils durch eine Firewall gegeneinander geschützt. Ein nennenswerter Service, der in der DMZ zur Verfügung gestellt wird, um die externen IAP-Netzwerke mit dem in Kühlungsborn zu verbinden, ist das Virtual Private Network (VPN). Durch diese Netzwerktechnologie ist es möglich, lokale Netzwerke über das Internet räumlich unabhängig voneinander zu erweitern. Ein Beispiel dafür ist, dass die Steuerrechner in Andenes in Norwegen mittels des VPN virtuell genauso erreichbar sind als würden sie im Hauptgebäude des IAP in Kühlungsborn stehen. Praktisch ist es damit auch möglich z.B. Simulationsläufe auf dem Parallelrechner von Zuhause aus zu überwachen. Ein VPN ist ein gesichertes Netzwerk, das auf einem ungesichertem Übertragungsmedium basiert [5]. Allgemein bieten VPN's eine aktive Form von Sicherheit durch Datenverschlüsselung, Datenkapselung oder einer Kombination aus beiden [6].

2.2. Zentrale Datenarchivierung

Dem IAP steht ein zentraler Fileserver zur Verfügung um die anfallenden großen Datenmengen speichern zu können. Für Sicherheitskopien (Backups) der wichtigsten Daten, der Mitarbeiter PC's, ist ein Backupserver vorhanden. Auf ihm können auch Daten gespeichert werden die innerhalb des IAP's für alle Mitarbeiter verfügbar sein sollen.

2.2.1. Technische Details des Fileservers

Der Fileserver besteht aus einem Origin® 350 Steuerrechner von SGI® und einer Scalar i2000 Magnetbandbibliothek von Quantum. Der Steuerrechner von SGI® besitzt acht R16000 Prozessoren mit je 700 MHz. Sein Betriebssystem mit dem Namen Data Migration Facility (DMF), von dem auch die allgemeine Bezeichnung des Fileservers (im folgenden: DMF) abgeleitet ist, verwaltet die Daten auf der Magnetbandbibliothek. Die Bandbibliothek kann ein Datenvolumen von 400 TB aufnehmen. Die Daten werden jedoch aus Datensicherheitsgründen meist gespiegelt, also doppelt abgespeichert. Somit stehen 200 TB für die Datensicherung zur Verfügung. Weiterhin

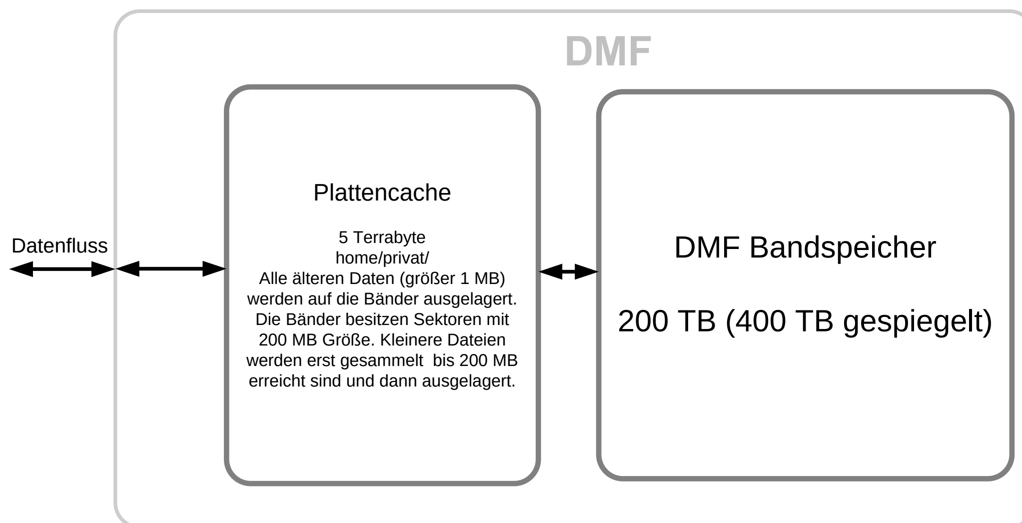

Abbildung 2.2.: Schematische Darstellung des zentralen Fileservers des IAP's.

besitzt der DMF einen Plattencache mit ca. 5 TB Speicherkapazität gespiegelt (s. Abb. 2.2). Der Bereich des DMF, der das Datenarchiv darstellt (`/home/private/..`), steht allen Linux-Netzwerkservern als gemounteter Ordner zur Verfügung. Von den Arbeitsplatzrechnern kann der DMF als Netzlaufwerk eingebunden werden. Für Backups der Mitarbeiter PC's steht ein Backupserver zur Verfügung (s. Abb. 2.1). Auch er kann als Netzlaufwerk auf den Desktop PC's eingebunden werden. Hier kann auf den Teil „`/iappool/..`“ zugegriffen werden. Dieser Teil des Backupservers enthält zum Beispiel Ordner wie „`/public`“ oder „`/scratch`“, die permanent für den Datenaustausch zwischen den Mitarbeitern des IAP's zur Verfügung stehen sollen. Bei der Verwendung des Fileservers, sind ein paar Dinge zu beachten, die das Arbeiten mit dem Fileserver erleichtern, bzw. die Lebensdauer des Servers beeinflussen können. Für das bessere Verständnis soll daher kurz auf die Technik der Bandbibliothek eingegangen werden.

Die Daten werden auf die Magnetbänder mittels des Linear Tape Open (LTO) Verfahren [7] geschrieben. Die Bibliothek besteht aus einer großen Anzahl von Bändern, die an einem festen Platz in einem Regal gelagert werden. Ein vorhandener Roboterarm dient dazu ein angefordertes Band aus dem Regal zu nehmen, es in ein Lesegerät einzulegen und es nach beendetem Lesevorgang wieder an seinen Platz im Regal zu deponieren. Die Daten auf einem Magnetband können nicht wie auf einer CD oder Festlatte ausgelesen werden indem vom Lesekopf einfach der entsprechende Sektor angesteuert wird. Das Magnetband muss eventuell bis zum letzten Eintrag durchgespult werden an dem die gewünschten Daten gespeichert sind. So ist die Zugriffszeit auf Daten zum Teil davon abhängig wo diese auf einem Band gespeichert sind. Hierbei kann es nach Herstellerangaben bis zu einigen Minuten dauern, bis die Daten verfügbar sind. Dies ist aber nicht der Regelfall, sondern eher die Ausnahme. Die Bänder sind in Sektoren von 200 MB Größe aufgeteilt. Zu jeder Datei werden vom migrierenden Filesystem zusätzliche Daten auf den Bändern gespeichert. Diese zusätzlichen Daten nehmen natürlich auch zusätzlichen Speicherplatz auf dem Band ein, es entsteht so genannter Overhead. Dieser Overhead wird im Verhältnis zur Datei größer, je kleiner diese ist. Die auf dem DMF gespeicherten Dateien sollten idealer Weise eine Größe zwischen 10 MB und 20 GB besitzen, um die verfügbare Speicherkapazität und Performance optimal nutzen zu können.

2.2.2. Funktion des Fileservers

Die Bandbibliothek ist so konzipiert, dass sie dem Anwender die Daten mit so geringen Zugriffszeiten wie möglich zur Verfügung stellt. Das wird unter anderem mit hohen Umdrehungen beim Auslesen des jeweiligen Bandes realisiert. Ein Vorgang der den Verschleiß der Bandbibliothek im hohen Maße beschleunigt ist, wenn viele kleine Dateien einzeln ausgelesen werden. Zum besseren Verständnis soll der mechanische Vorgang kurz erläutert werden.

Der DMF erhält den Befehl eine Datei (hier angenommen 2 MB) auszulesen. Daraufhin holt der Roboterarm das entsprechende Magnetband aus seinem Schacht und legt es in das Lesegerät ein, wie schon beschrieben. Dann spult das Lesegerät das Band bis zu dem Sektor, in dem die Datei abgelegt ist und liest sie aus. Nach Abschluss des Lesevorganges wird das Band wieder auf die Anfangsposition zurückgespult und der Roboterarm legt das Band wieder in dessen Schacht zurück. Gleich danach erhält der DMF den Befehl die nachfolgende Datei auszulesen. Jetzt passiert genau das Selbe wie zuvor bis auf, dass das Lesegerät einen einzigen Dateieintrag weiterspulen muss. Wenn es sich um ein größeres Dateiarchiv handelt, kann sich dieser Vorgang um die tausendmal wiederholen, dass für die Mechanik des DMF und das Band eine große Belastung darstellt. Wenn viele kleine Dateien auf dem DMF abgelegt werden kann es passieren, dass sie nicht zusammenhängend auf ein Band geschrieben werden, so dass zwischen den einzelnen Dateien andere Daten liegen. Gibt ein Benutzer dem DMF den Befehl diese Dateien alle mit einem Mal auszulesen, kommt das Lesegerät in einen permanenten Start- Stopp- Zustand, weil es bei den Daten, die zwischen den auszulesenden Dateien in einen Spulmodus wechselt und bei den auszulesenden wieder in einen Lesemodus. Auch dieser Vorgang belastet den DMF sehr stark. Wenn diese Arten des Datenauslesens zusätzlich von mehreren Benutzern gleichzeitig praktiziert werden, verlängern sich die Zugriffszeiten durch die häufigen Bandbewegungen, schnell so sehr, dass der Datenverkehr zusammenbricht. Ein effektiveres Arbeiten mit dem DMF wird erreicht, in dem erstens die Dateigrößen angepasst und zweitens benötigte Dateien möglichst immer zusammenhängend ausgelesen werden. Wie das erreicht werden kann, wird nachfolgend erklärt.

Müssen aus verschiedenen Gründen viele kleine Dateien gespeichert werden, können sie in einer Archivdatei, z.B. einem RAR-Archiv, zusammengefasst werden. Damit der Fileserver mehrere Zugriffe gleichzeitig verarbeiten kann, werden die Daten

bei Lese- oder Schreibanfragen immer erst auf einen ca. 5 TB großen „Plattencache“ (s. Abb. 2.2), also einem Zwischenpuffer geschrieben. Von dort aus werden die Daten für den jeweiligen Benutzer über das Netzwerk verfügbar gemacht, bzw. auf die Bänder ausgelagert. Wenn die Daten auf dem Plattencache vorliegen, sollte quasi keine Verzögerung bei einem Datenzugriff von einem PC aus vorliegen. Der Hersteller gibt die Verzögerungszeit mit unter einer Sekunde an; der Benutzer am PC nutzt das Netzlaufwerk in diesem Fall als wäre es direkt auf seiner Festplatte. Da die Datenkapazität des Plattencaches im Gegensatz zur Bandbibliothek sehr begrenzt ist, werden die Daten mittels zwei Kriterien hierarchisch eingestuft, nach denen sie dann auf dem Plattencache gelöscht bzw. auf Band ausgelagert werden. Dies ist zum Einen davon abhängig wie lange der letzte Aufruf der Daten auf dem DMF zurück liegt und zum Anderen von der Größe der einzelnen Dateien. Generell ist der DMF so konfiguriert, dass einmal aufgerufene Daten für 24 Stunden auf dem Plattencache vorliegen. Wenn der Plattencache durch viele Zugriffe vollläuft, müssen Dateien ausgelagert werden; hier stehen die kleinsten Dateien in der Hierarchie ganz oben, sodass die größten als erstes wieder auf die Bänder ausgelagert werden. Danach werden die Dateien über ihre Lebensdauer auf dem Plattencache priorisiert, d.h. die ältesten Dateien werden zuerst ausgelagert. Hierbei ist besonders zu beachten, dass Dateien, die kleiner 1 MB sind, nie auf Band ausgelagert werden. Daraus folgt, dass so kleine Dateien für die Dateiarchivierung auf dem DMF immer als Verbund von mehreren Dateien in ein Archiv gepackt werden müssen, da sonst das Arbeitsvolumen des Plattencache massiv verkleinert wird.

2.2.3. Praktisches Arbeiten mit dem Fileserver

Die verschiedenen möglichen Zustände, die die Daten auf dem DMF annehmen können, sind in der Tabelle 2.1 aufgeführt. Für das optimierte Arbeiten mit dem DMF können diese Zustandsbezeichnungen unter Linux in Verbindung mit Befehlen aus der Tabelle 2.2 verwendet werden. Diese Befehle können in eigenen programmierten Routinen, in Shellskripten oder direkt in der Konsole angewendet werden. Sollte z.B. das Laden von Dateien merklich zu lange dauern, kann mit der Unix- Befehlszeile [8] „`df -h`“ der Zustand des Plattencaches überprüft werden. In Abb. 2.3 ist in der

Zustand	Bedeutung
REG	Daten auf Disk
OFL	Daten auf Band
DUL	Daten auf Band und Disk
MIG	Daten werden von Disk auf Band kopiert
UNM	Daten werden von Band auf Disk kopiert
PAR	Daten beim Kopieren von Band auf Disk, teilweise schon auf Disk

Tabelle 2.1.: Übersicht der möglichen Dateizustände auf dem DMF.

Befehl	Auswirkung
dmls	Statusabfrage
dmget	Dateien werden vom Zustand OFL → DUL gebracht
dmgetdir	Ordner werden vom Zustand OFL → DUL gebracht
dmput(-r)	Dateien werden vom Zustand REG → DUL(OFL) gebracht
dmputr	Dateien werden vom Zustand DUL → OFL gebracht
dmputdir(-r)	Ordner werden vom Zustand REG → DUL(OFL) gebracht
dmputrdir	Ordner werden vom Zustand DUL → OFL gebracht

Tabelle 2.2.: Übersicht DMF- Befehle.

markierten Zeile der Zustand des Plattencaches aufgeführt. Dort sind nacheinander die aktuellen Eigenschaften angegeben: Größe insgesamt in Terrabyte, verwendeter Speicher in Terrabyte, freier Speicher in Gigabyte, verwendete Speichermenge in Prozent und der Pfad unter dem der DMF gemountet ist. Laut Herstellerangaben hält das Migrationssystem des DMF's zwischen 50 % und 80 % des verwendeten Plattencaches frei. Bei einer Speicherbelegung von über 80 % wird es aktiv und beginnt den Plattencahe frei zu räumen. Dabei kann es passieren, dass sich die Zugruffszeiten auf den DMF verlängern. Praktische Erfahrungen haben gezeigt, dass der DMF sehr langsam arbeitet, sobald die Nutzung über 74 % steigt. Trifft dies einmal zu und wird mit der oben beschriebenen Befehlszeile bestätigt, sollte der Benutzer sich zuerst fragen: "Welche Daten habe ich in der letzten Zeit vom DMF verwendet?". Der Nutzer kann nun überprüfen, ob diese Daten vielleicht nach ihrer Verwendung nicht vom Plattencache gelöscht wurden. Das ist möglich indem das Verzeichnis über dem

```
user@rechner:~> df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/system-root_lv
                      4.0G  407M  3.7G  10% /
udev                  7.9G  148K  7.9G   1% /dev
/dev/sda3             204M   81M  123M  40% /boot
/dev/mapper/system-opt_lv
                       48G  8.5G   40G  18% /opt
/dev/mapper/system-tmp_lv
                      3.0G  472M  2.6G  16% /tmp
/dev/mapper/system-usr_lv
                       17G  4.2G   13G  26% /usr
/dev/mapper/system-var_lv
                       32G  712M   32G   3% /var
/dev/mapper/system-home_lv
                      1.8T  1.3T  587G  69% /home
fab1:/dmf/perm/home   1.9T  1.2T  789G  59% /home/dmf
orion:/home           1.9T  247G  1.6T  14% /home/orion
user@rechner:~> 
```

Abbildung 2.3.: Die Markierung zeigt die Eigenschaften des DMF-Plattencache.

```
user@rechner:~/dmf/LIMA/LIMA-DATA/global/1971> du * -sh
14G     01
8.0K    02
8.0K    03
8.0K    04
8.0K    05
8.0K    06
8.0K    07
8.0K    08
8.0K    09
8.0K    10
8.0K    11
8.0K    12
user@rechner:~/dmf/LIMA/LIMA-DATA/global/1971> 
```

Abbildung 2.4.: Überprüfen des Datenvolumens von Ordnern auf dem DMF-Plattencache.

Verzeichnis mit den verwendeten Daten aufgerufen wird. Von dort aus kann mit dem Unixbefehl
„`du * -sh`“ abgefragt werden welcher der Unterordner wie viel Speicher auf dem Plattencache belegt. In Abb. 2.4 wird gezeigt, dass der Unterordner „`01`“ des Ordners „`1971`“ 14 GB Speichervolumen auf dem Plattencache belegt. Wenn das ein Ordner ist, auf den der Nutzer zuletzt zugegriffen hat, kann er die Daten über die Konsole wieder in den Status OFL (s. Tab. 2.1) bringen, um den Speicher auf dem Plattencache wieder freizugeben. Wurden die Daten dieses Ordners vom Nutzer nicht verwendet, sollten sie nicht eigenmächtig von ihm in den Zustand OFL gebracht werden, da sie möglicherweise zur Zeit von einem anderen Mitarbeiter des

```
user@rechner:~/dmf/LIMA/LIMA-DATA/global/1971> dmfind ./ -state "DUL"
./01/19710129.18.bin
./01/19710130.00.bin
./01/19710130.06.bin
./01/19710130.12.bin
./01/19710130.18.bin
./01/19710131.00.bin
./01/19710131.06.bin
./01/19710131.12.bin
./01/19710131.18.bin
user@rechner:~/dmf/LIMA/LIMA-DATA/global/1971>
```

Abbildung 2.5.: DMF-spezifische Statusabfrage von Dateien auf dem Plattencache. Es werden alle Dateien mit zugehörigem Unterverzeichnispfad angezeigt, die sich im Verzeichnis „`1971`". befinden und den Status DUL besitzen (s. Tab. 2.1).

IAP verwendet werden. In so einem Fall sollte erst Rücksprache mit den Mitarbeitern des IAP gehalten werden, die Zugriff auf die betroffenen Daten haben. Zur Identifizierung der Daten, die im Zustand DUL (s. Tab. 2.1) sind, kann die Befehlszeile „`dmfind ./ -state "DUL"` " verwendet werden (s. Abb. 2.5). Sind die Daten nicht in der Benutzung von anderen Mitarbeitern des IAP, können sie mit der Befehlszeile „`dmfind ./ -state "DUL" | dmput -r` " in den Zustand OFL gebracht werden. In diesem Fall dauert die Aktion nur wenige Sekunden, weil die Daten nur von dem Plattencache gelöscht werden, da sie schon auf Band gespeichert sind. Der Zustand der Daten kann nachträglich noch einmal mit „`du * -sh`" und der des DMF's mit „`df -h`" überprüft werden.

3. Großrechentechnik am IAP

Im folgenden Kapitel werden die dem IAP zur Verfügung stehenden Computserver beschrieben. Es wird auf die Unterschiede der Hardware und Software eingegangen. Außerdem wird die optimale Nutzung der Computserver anhand von praktischen Beispielen erläutert.

3.1. Hardware und Software

Dem IAP stehen drei Mehr-Prozessorrechner für je eine wissenschaftliche Abteilung, der Vektorrechner NEMO und der Parallelrechenserver Orion zur Verfügung.

Die Abteilungsrechner sind alle Power Edge 2900™ Modelle von Dell® und stehen im Hauptgebäude (s. Abb. 2.1). Sie besitzen alle die gleiche Hardwarekonfiguration. Es sind jeweils zwei Intel® Xenon™ X86_64-Vierkernprozessoren (Quadcore CPU's), 8 GB Hauptspeicher und 2 TB Festplattenspeicher verbaut. Die Prozessoren mit dem Codenamen Woodcrest arbeiten mit einem Takt von 2,6 GHz, sie sind 64 Bit-CPU's, die den „allgemein“ gebräuchlichem x86-Prozessor-Befehlssatz besitzen. Das garantiert eine hohe Softwarekompatibilität gegenüber dem Orion. Neben dem Betriebssystem Suse Linux Enterprise 10 ist Software wie z.B. MATLAB, IDL, PV-Wave, grads und convert installiert. Weitere Software ist je nach Bedarf der einzelnen Abteilung installiert. MATLAB und IDL laufen auf den Abteilungsservern auch im 64 Bit-Modus. Des Weiteren fungieren sie für die jeweiligen Abteilungen als Mailserver.

Orion ist ein Altix450™ -Server von SGI® und besitzt 32 Dualcore Prozessoreinheiten (CPU's), also logisch 64 Prozessoren. Die Dualcore-CPU's mit der Modellbezeichnung Itanium2™ von Intel® arbeiten mit einem Takt von 1,6 GHz. Sie basieren auf der 64 Bit Architektur IA64 und besitzen 18 MB Cache. Der Codename der CPU's

ist Montecito. Der Vorteil von Prozessoren, die auf einer 64 Bit-Architektur basieren gegenüber den 32 Bit-Prozessoren ist, dass sie eine doppelt so große Speicheranbindung besitzen und so die Arithmetisch Logische Einheit (ALU) der CPU 64 Bit mit einem Takt verarbeiten kann. Die IA64-CPU's zeichnen sich gegenüber anderen x86[1] -64 Bit-CPU's durch eine sehr hohe Vektorperformance aus. Dies wird z.B. durch die 128 (64 Bit) Integer- und 128 (82 Bit) Gleitkomma-Register sowie 64 (1 Bit) Vorhersage-Register und weitere Spezial-Register erreicht. Dadurch können mehr Informationen in den Registern gehalten werden, ohne jedes Mal den langsamen Weg über Cache oder Arbeitsspeicher zu gehen, wenn Daten benötigt werden. Die IA64-Architektur verfügt weiterhin über einen großen CPU-Befehlssatz mit hoher Komplexität. So gibt es z.B. besondere Prozessorbefehle für aufwendige Gleitkommaoperationen. Orion verfügt über insgesamt 192 GB Hauptspeicher und 2 TB Festplattenplatz der durch Quotierung auf jeweils 10 GB pro Nutzer standardmäßig aufgeteilt ist. Dies kann mit dem Befehl „`quota`" nachvollzogen werden. Das Betriebssystem ist Suse Linux Enterprise Server 10. Der Computserver ist insbesondere für rechenintensive Berechnungen vorgesehen und optimiert. Als Compiler stehen der Intel Fortran (ifort) Compiler und für c bzw. c++ der GNU c++ Compiler (gcc vers. 4.1.2) zur Verfügung. Zusätzlich können die Intel Mathematic Kernel Library™ (IMKL)[2] und die Numerical Algorithms Group Library™ (NAG)[3] genutzt werden. Als Hilfsmittel zur Analyse und Optimierung stehen der Intel Debugger™ (idb), der Gnu Debugger (gdb), der Intel Thread Profiler™ , die Intel® Integrated Performance Primitives Bibliothek (ipp) und der VTune™ Performance Analyzer von Intel® zur Verfügung. Um Rechenaufgaben voneinander und vom Betriebssystem rechentechnisch zu trennen, werden sie auf dem Orion in so genannte *Cpusets* und *Vnodes* eingeteilt. Ein *Vnode* ist ein virtueller Knoten, der die tatsächliche Hardware des Rechners widerspiegelt. Orion besteht aus einer Reihe von Modulen (Einschübe - Blades), die jeweils aus zwei Dual-Core-Prozessorschips mit je 3 GB RAM je CPU bestehen (s. Abb. 3.1a). Das heißt, ein *Vnode* entspricht hier einem Blade mit vier CPU's und 12 GB Speicher. Orion besteht

[1] x86 bezeichnet den Befehlssatz einer von der Firma Intel entwickelten Mikroprozessor-Architektur

[2] Die Intel MKL beinhaltet Funktionen der linearen Algebra (BLAS, LAPACK, DSS), diskrete Fourier Transformationen (DFT), Vektor Mathematik (VML) und vektor-statistische Zufallsgeneratoren. Informationen unter: www.intel.com/cd/software/products/asmo-na/eng/307757.htm

[3] NAG Numerical Libraries ist eine umfassende Software-Unterprogrammbibliothek für numerische und statistische Problemstellungen. Informationen unter: www.nag.co.uk

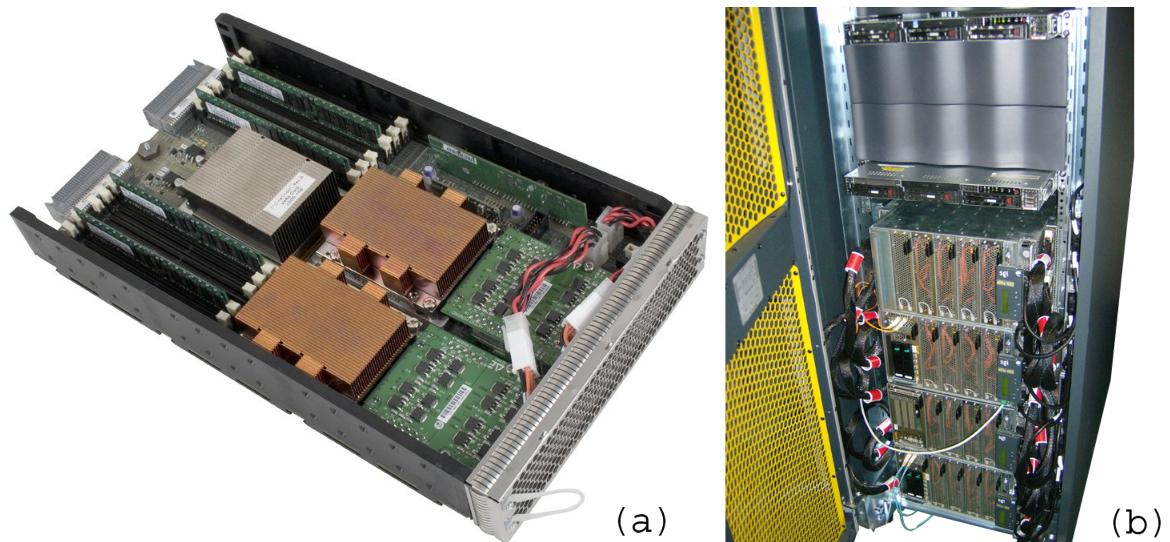

Abbildung 3.1.: In der Abbildung (a) ist ein Einschub mit zwei Dual-Core-Prozessoren und dem dazugehörigem Hauptspeicher des Orion (b) zu sehen.

also aus einem Gehäuse mit 16 Einschüben (s. Abb. 3.1b). Die parallele Architektur des Orion basiert auf der sog. NUMA (Non-Uniform Memory Access) Architektur [9]. Die NUMA-Architektur liegt, wenn sie den Paralleltechniken „System mit gemeinsamen Speicher“[4] oder „System mit lokalem bzw. verteiltem Speicher“[5] zugeordnet werden soll, zwischen diesen (s. Abb. 3.2(a)). Die Idee dieser Architektur ist die Simulation eines gemeinsamen Speichers mit globalem Adressraum auf Systemen mit verteiltem Speicher. Die Vorteile dieser Architektur liegen im ausgewogenen Verhältnis von einfacher Programmierung und der guten Systemerweiterbarkeit bis ca. 1024 CPU's. Die unkomplizierte Programmierung in diesem Parallelsystem ist durch die dynamische Verwaltung der Orion Ressourcen mittels des sog. Queuesystems PBSpro gegeben. Der lokale Kommunikations-Controller Shub 2.0 (s. Abb. 3.2(b)) entscheidet, ob ein (schneller) lokaler Speicherzugriff erfolgt oder ein (langsamerer) Nachrichtenaustausch mit einer entfernten CPU durchgeführt wird. Bei Bedarf werden nichtlokale Daten in die Prozessor-Caches geladen. Eine hohe Skalierbarkeit der Rechenleistung wird durch Cache-Kohärenz (ccNUMA - *cache coherent Non-Uniform*

[4] engl. shared memory
[5] engl. local memory bzw. distributed memory

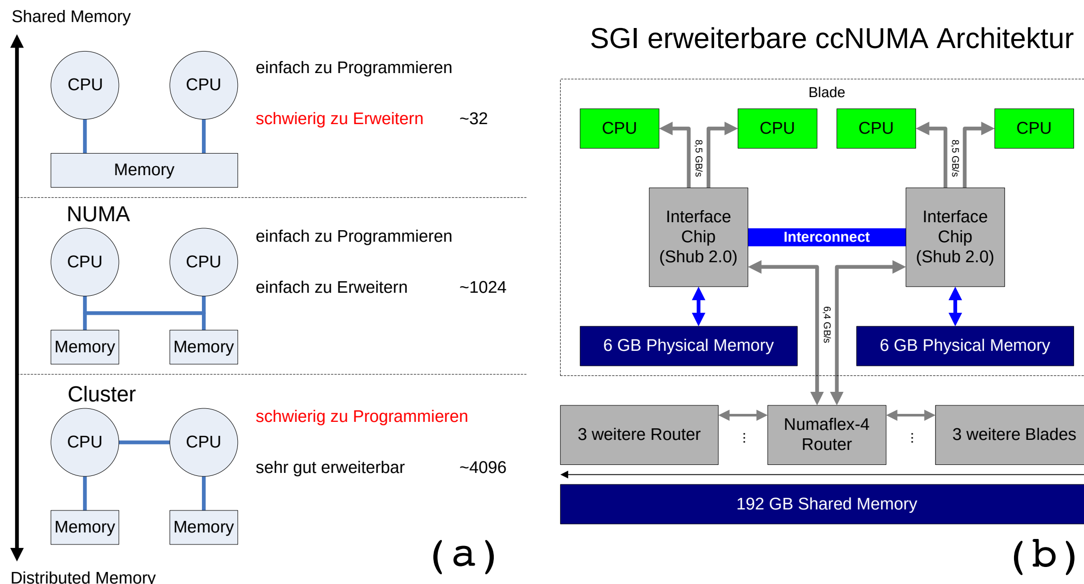

Abbildung 3.2.: In der Abbildung (a) werden verschiedene Parallelarchitekturen mit Vor- und Nachteilen gezeigt, die in, bzw. zwischen den Bereichen der technischen Realisierungen Shared Memory und Distributed Memory liegen. In (b) ist die erweiterbare ccNUMA Architektur von SGI, speziell für die Ausführung des Orion zu sehen.

Memory Access) erreicht. Cache-Kohärenz bedeutet in diesem Fall, dass ein Datenelement, das in den Caches von verschiedenen Prozessoren liegt, die gleichen Werte enthalten muss [10]. Realisiert wird die Cache-Kohärenz durch Verzeichnisse, die jeder Knoten besitzt, in denen festgehalten wird, wo sich Kopien von lokalen Speicherblöcken befinden. Weitere Informationen zu diesem Thema sind in [9] oder [11] zu finden. Neben den beiden Itanium2-CPU's und den 12 GB RAM gehören zu einem Knoten auch die Shub-Controller, die die CPU's mit dem RAM und dem Netzwerk koppeln. Sie realisieren die Verzeichnis-basierte Cache-Kohärenz direkt in der Hardware. Benachbarte Knoten werden über die Numaflex-4 Router mit 6,4 GB/s gekoppelt, die das interne Netzwerk des Orion darstellen (s. Abb. 3.2(b)). Die *Cpusets* werden aus den *Vnodes* gebildet. Ein *Cpuset* ist eine logische Zusammenfassung von CPU's und Speicher, der *Vnodes*. Auf Orion gibt es zwei Arten von *Cpusets*, statische und dynamische. Das erste *Cpuset* ist statisch festgelegt und steht ausschließlich dem Betriebssystem und den interaktiven Anwendungen zur Verfügung (*Bootcpuset*). Es besteht aus 4 CPU's mit 12 GByte Hauptspeicher. Die restlichen *Vnodes* wer-

den ausschließlich vom Batchsystem (Queuesystem PBSpro) dynamisch verwaltet. PBSpro legt entsprechend den Anforderungen der Batchjobs in den Queues *Cpusets* an. Minimal kann ein *Cpuset* aus einer CPU mit 100 MB Speicher bestehen. Der Vorteil dieser Konfiguration ist, dass Programme im Batchsystem nicht von Aktivitäten des Betriebssystems gestört werden können. Programme, die im Batchsystem ein *Cpuset* zugewiesen bekommen haben, können sich auch untereinander nicht beeinflussen. Ist ein *Cpuset* größer als ein *Vnode* (mehr als 4 CPU's), werden die direkten Nachbarblades verwendet und intelligent verwaltet.

3.2. Praktisches Arbeiten mit der Großrechentechnik

Im folgenden Abschnitt wird eine Einführung in das praktische Arbeiten mit dem Parallelserver Orion und den Abteilungsrechnern gegeben. Dabei wird auf Beispielskripte und Beispielprogramme in Fortran eingegangen.

3.2.1. Orion

Der Orion-Parallelserver ist, wie schon erwähnt, durch seine Hardware auf Berechnungen numerischer Problemstellungen mit C, C++ oder Fortran ausgerichtet. Programmierumgebungen wie z.B. für IDL sind auch installiert, können aber nur im 32-Bit Betrieb emuliert werden, da die spezielle Prozessorarchitektur des Orion von den meisten Softwareherstellern nicht unterstützt wird. Somit ist der Einsatz dieser Software auf dem Orion nur bedingt empfehlenswert. Da der überwiegende Teil der Programme für den Orion in Fortran90 geschrieben ist, ist auch das folgende Beispielprogramm in Fortran90 geschrieben. Für die Kompilierung der Programme werden Intel® -Compiler (hier ifort) verwendet, weil sie für Itanium2-CPUs einen effizienteren Code erzeugen als andere Compiler. Kleinere Anwendungen, die keine Parallelisierung mit mehr als zwei Prozessoren voraussetzen, sollten besser auf Abteilungsservern ausgeführt werden. Die Parallelisierung von Programmen kann durch die Nutzung des Message Passing Interface (MPI) Standards, Open specifications for Multi-Processing (OpenMP) bzw. Autoparallelisierung durch den Compiler erfolgen.

Weil OpenMP für NUMA-Rechnerarchitekturen entwickelt wurde, wird es zur parallelisierten Programmentwicklung auf dem Orion empfohlen.

Da die parallelisierten Programme in einem Shell-Skript gestartet werden, ist es sinnvoll Makefiles anzulegen, in denen die Programme kompiliert werden. Codebeispiel 3.1 zeigt den Quellcode eines Makefiles. In Zeile zwei sind die Compileranweisungen enthalten, wobei „`ifort`“ den Compiler aufruft und „`-openmp`“ die Parallelisierung des Programms mit OpenMP ermöglicht. Weiter bezeichnet „`bsp.f`“ die Fortrandatei und „`testprog`“ die ausführbare Binärdatei, die erstellt wird. Eine Compileranweisung, die eventuell nützlich sein kann, wenn die Ausgabewerte auf einem anderen Rechner, dessen Betriebssystem mit der „Big Endian“ Formatierung arbeitet, weiterverarbeitet werden müssen, ist „`-convert big`“. Die Anweisung „`-O2`“ ist eine Optimierungsanweisung für den Compiler. Es sind insgesamt vier „`-O`“ Optimierungsanweisungen verfügbar, „`-O0`“ bis „`-O3`“. Welche Auswirkungen diese und andere Compileranweisungen auf das Programm haben ist in den Man-Pages mit dem Aufruf „`man ifort`“ abrufbar. Wichtig ist zu wissen, dass die Standardoptimierung „`-O2`“ ist, auch ohne dass sie in der Kompilieranweisung angegeben wird. Bei „`-O3`“ wird aggressiv auf kurze Programmlaufzeiten optimiert, dabei wird z.B. die Genauigkeit von Floating-Point Operationen heruntergesetzt, was zu ungenauen Ergebnissen führen kann. Dem kann wiederum mit der Anweisung „`-fp-model source`“ entgegengewirkt werden. Das Makefile muss den Namen „`makefile`“ tragen damit es mit dem Befehl „`make`“ in der Konsole ausgeführt werden kann.

Listing 3.1: Beispiel: makefile

```
1 testprog: bsp.f
2         ifort -openmp -O2 -o testprog bsp.f
```

Auf dem Orion ist ein so genanntes PBS Queue-System installiert um benötigte Ressourcen für verschiedene Rechenaufgaben gewährleisten zu können. So werden durch das Queue-System genau die Ressourcen, wie z.B. Prozessoren, Hauptspeicher und Rechenzeit, zur Verfügung gestellt, die für die Rechenaufgaben vorher im Queue-Skript vereinbart wurden. Außerdem gerät das System nicht in einen Überlastungszustand, so dass sich z.B. mehrere Prozesse eine CPU teilen müssen oder der Hauptspeicher nicht ausreicht, weil zu viele Rechenaufgaben gleichzeitig bearbeitet werden. Es wird daher empfohlen alle rechenintensiven Programme im PBS

Queue	Memory	Walltime	Priorität
q4s	12 GB	2 h	100
q8s	24 GB	2 h	95
q16s	48 GB	2 h	90
q4l	12 GB	8 h	80
q8l	24 GB	8 h	75
q16l	48 GB	8 h	70
q4xl	12 GB		60
q8xl	24 GB		55
q16xl	48 GB		50
q32xl	64 GB		45

Tabelle 3.1.: Mögliche Queue's auf dem Orion. Die maximal anforderbare CPU Anzahl steht jeweils hinter dem „`q`". Die maximale Walltime (Rechenzeit in Realzeit) ist durch s (short), l (long), xl (extrem long) gekennzeichnet. Die Queues mit der höchsten Priorität werden vorrangig behandelt.

Queue-System abzuarbeiten. Im interaktiven Betrieb, also Programme die nicht im Queue-System laufen (laufen auf dem *Bootcpuset* s. Kap 2.1), sind nur kleinere Anwendungen wie z.B. editieren, kompilieren, debuggen usw. erlaubt. Das bedeutet konkret, dass interaktive Benutzerschnittstellen des Linux Betriebssystems, wie z.B. der Konquerror, nur von den Abteilungsservern aus auf dem Orion zu benutzen sind. Andernfalls ist die Gesamtlaufzeit einer Rechenaufgabe durch die User-limits auf 10 min. begrenzt. Es sind auf Orion zehn Queues installiert (s. Tab. 3.1). Durch sie werden die maximal nutzbaren Ressourcen Prozessoren, Hauptspeicher und Rechenzeit (Walltime) begrenzt. Die voreingestellte Queue ist die „`q4s`". Ist der Orion aktuell soweit ausgelastet, dass die benötigten Ressourcen der ausstehenden Rechenaufgaben zur Zeit nicht verfügbar sind, werden sie solange in der Warteschlange des PBS-Queue-Systems gehalten, bis die ihnen zugesicherten Ressourcen zur Verfügung stehen. Wie viele und welche Queues aktuell auf dem Orion abgearbeitet werden, bzw. in der Warteschlange stehen kann mit der Anweisung „`qstat -q`" abgefragt werden. Durch die Eingabe der Anweisung „`qmgr -c "print server"`" wird eine Auflistung aller Parameter jeder Queue ausgegeben, mit „`qstat -Qf`" der Status der Queues. Die Abarbeitungspriorität der unterschiedlichen Queues, richtet sich nach dem jeweiligen Ressourcenanspruch (s. Tab. 3.1), so dass die Queues mit dem geringeren Res-

sourcenanspruch vorrangig behandelt werden. Ein Programm kann nur über ein so genanntes Queue-Skript in das PBS-Queue-System eingereiht werden. Mit der Anweisung „`qsub`“ wird das Queue-Skript dann direkt in der Konsole (Shell) abgesetzt. Dies kann geschehen indem benötigte Steuerparameter mit in die „`qsub`“ -Anweisung geschrieben werden, wie z.B.: „`qsub -q q8s -l ncpus=8 -l mem=2gb skriptname`“. Da für die Laufzeit der Programme in der Queue sehr oft noch zusätzliche Umgebungsvariablen gesetzt werden müssen, wird empfohlen alle Anweisungen zusammengefasst in das Queue-Skript zu schreiben. Die Anweisung reduziert sich dann auf „`qsub skriptname`“. Ein Beispielskript mit den wichtigsten Anweisungen der Umgebungsvariablen für das Programm sowie seiner Kompilierung wird im Quellcode 3.2 gezeigt. In der ersten Zeile des Beispielskriptes wird darauf hingewiesen, dass es sich um ein Bourne-Shell-Skript („`sh`“) handelt. In den Zeilen zwei und drei folgen die Anweisungen für das PBS Queue-System. Mit „`-q`“ wird die gewünschte Queue festgelegt (s. Tab.3.1). Wird diese Option weggelassen, ist automatisch die Standard-Queue „`q4s`“ festgelegt. Die Option „`-l`“ fordert die Hardwareressourcen für die Queue an. Werden die Ressourcen nicht explizit angefordert,

Listing 3.2: Beispiel: Queuescript

```
1  #!/bin/sh
2  #PBS -q q4xl
3  #PBS -l select=1:ncpus=4:mem=6gb
4
5  cd /home/oa030/fortran/programme
6  make
7
8  ulimit -s unlimited
9  export KMP_AFFINITY=disabled
10 export OMP_NUM_THREADS=4
11 export KMP_LIBRARY=turnaround
12 export KMP_MONITOR_STACKSIZE=512MB
13 export KMP_STACKSIZE=512MB
14 dplace -x2 -c 0-3 ~/fortran/programme/testprog
```

werden sie mit den Standardeinstellungen, eine CPU und 200 MB Hauptspeicher, festgelegt. In dem Beispiel werden in Zeile drei mit „`select`“ die Anzahl der *Cpusets*, mit „`ncpus`“ die maximale Anzahl der CPU's und mit „`mem`“ die maximale Größe des Hauptspeichers angefordert. Dabei ist zu beachten, dass die Anzahl der CPU's und die Größe des Hauptspeichers nicht die Maximalwerte der gewählten Queue überschreitet (s. Tab.3.1). In den Zeilen fünf und sechs wird die Kompilierung des Programms vorgenommen. Hier wird zunächst in das Verzeichnis des Makefiles gewechselt und anschließend mit dem Befehl „`make`“ kompiliert. Da das Fortran Programm mittels OpenMP parallelisiert werden soll, folgen in den Zeilen acht bis dreizehn Anweisungen, die standardmäßige Limitierungen seitens OpenMP ersetzen bzw. den Grad der Parallelisierung beeinflussen. Die standardmäßig vom System zur Verfügung gestellten Ressourcen und Limitierungen können mit dem Kommando „`ulimit -a`“ abgefragt werden. Mit der Anweisung „`ulimit -s unlimited`“ wird die Stackgröße für das gesamte Programm auf das mögliche Maximum gesetzt. Die Stackgröße kann auch explizit den Programmanforderungen angepasst werden in dem für „`unlimited`“ die gewünschte Speichergröße in kB, z.B. 1 GB mit „`1024000`“, angegeben wird.

Für die Möglichkeit der Optimierung des Ablaufes einzelner Programme wurde das Konzept der Threads (zu Deutsch: *Faden*) geschaffen. Während unter Multitasking das Nebeneinanderlaufen mehrerer verschiedener Programme zu verstehen ist, sind Threads die kleinsten Einheiten eines Programms, die in der Lage sind parallel zueinander zu laufen. Mit dem Konzept der Threads ist es möglich unabhängige Teile eines Programms parallel ablaufen zu lassen, wobei die Threads dem Shared Memory -Ansatz folgen, also den Arbeitsspeicher gemeinsam nutzen können. Die OpenMP-Parallelisierungstechnologie setzt an diesem Ansatz an und parallelisiert anhand von Threads.

Um bei der Verarbeitung großer Felder einen Abbruch des Programms durch eine „`segmentation fault`“ Fehlermeldung zu verhindern, sollte auch die Stackgröße der Threads erhöht werden, wie es im Beispiel, Zeile `13` zu sehen ist [12]. In der Zeile `12` wird mit „`KMP_MONITOR_STACKSIZE =512 MB`“ die Größe des sog. Monitor Threads festgelegt [13]. Dieser Thread wird von OpenMP während der Programmausführung benutzt um alle anderen Threads zu überwachen. Unter Umständen muss der Stack dieses Threads gegenüber der „`KMP_STACKSIZE`“ größer gewählt werden. OpenMP besitzt drei Ausführungsvarianten [14], die mit „`KMP_LIBRARY`“ gewählt werden kön-

nen. Der voreingestellte Standard ist „`throughput`“, er ist für Mehrprozessorsysteme entwickelt worden, deren Hardwareressourcenverteilung in einer Mehrbenutzerumgebung nicht explizit geregelt ist. In diesem Fall regelt OpenMP die Ressourcenverteilung dynamisch, sodass sie voll ausgeschöpft werden. Bei mehreren Benutzern gleichzeitig werden die Ressourcen wiederum den Rechenaufgaben angemessen verteilt, sodass alle Rechenaufgaben der Serverauslastung entsprechend am effektivsten verarbeitet werden. Diese Variante trifft auf den Orion-Server nicht zu, da die Ressourcenverteilung mittels des PBS-Queue-Systems in Verbindung mit „`dplace`“ im Vorhinein festgelegt wird. Auf dem Orion sollte für parallelisierte Rechenaufgaben immer die zweite Variante, wie im Skriptbeispiel in Zeile `11` zu sehen, „`turnaround`“ verwendet werden. Hier werden alle zur Verfügung stehenden Ressourcen für die effiziente Abarbeitung der Rechenaufgabe genutzt. Die dritte Variante wird mit dem Argument „`serial`“ festgelegt. Sie führt parallele Anwendungen in einem Thread aus, soll hier aber nicht weiter betrachtet werden. In der Zeile `10` wird mit „`OMP_NUM_THREADS`“ die Anzahl der Threads festgelegt, auf die OpenMP, den zu parallelisierenden Teil des Programms aufteilen soll. Ist diese Anweisung im Skript enthalten, muss die Anzahl der Threads im Fortran Quellcode mittels OpenMP-Anweisung nicht mehr angegeben werden. Idealer Weise wird jeder Thread auf einer CPU abgearbeitet, in unserem Beispiel also „`4`“. Mit der Anweisung „`KMP_AFFINITY`“ in Zeile `9` kann festgelegt werden wie OpenMP die einzelnen Threads an die verfügbaren CPU's bindet. Auf dem Orion-Server sollte die Option immer, wie im Beispiel ausgeschaltet werden, da diese Aufgabe durch die Anweisung „`dplace`“ in Zeile `14` wesentlich Hardwarenäher realisiert wird. Durch „`dplace`“ wird mit den angeforderten Hardwareressourcen der Queue vor dem Programmlauf ein virtueller Rechner erstellt. Es generiert ein *Cpuset*, das die gebildeten Threads während der Laufzeit fest auf den *Vnodes* und ihren CPU's hält. Dadurch wird ein Wandern der Threads auf CPU's anderer *Vnodes* und damit langsame Speicherzugriffe verhindert.

Das „`dplace`“ Kommando sollte immer im Batchbetrieb (also mit Queue-Skript) aber nie im interaktiven Betrieb verwendet werden! Dies könnte dazu führen, dass das *Bootcpuset* geteilt wird und dem Betriebssystem des Orion so nicht mehr die benötigten Hardwareressourcen zur Verfügung stehen, die es zum fehlerfreien Arbeiten benötigt.

Die Zeile `14` zeigt ein „`dplace`“ Kommando mit dem darauf folgendem Programmpfad der ausführbaren Binärdatei. Dem „`dplace`“ Kommando sind zwei Op-

tionen angehängt. Dabei ist „`-x2`“ auf dem Orion-Server immer mit anzugeben wenn mit OpenMP parallelisiert wird. Bei diesem Ausdruck handelt es sich um eine sog. *Skipmask*. Mit dieser Maske kann festgelegt werden, dass bestimmte Prozesse nicht an eine CPU gebunden werden. Das ist immer dann sinnvoll, wenn Applikationen Hilfsprozesse parallel laufen lassen, die sehr wenig CPU-Zeit benötigen. Im Beispiel wird mit „`-x2`“ verhindert, dass der Hilfsprozess (*monitor thread*) von OpenMP an eine CPU gebunden wird und diese blockiert. Das folgende Argument „`-c 0-3`“ übergibt die Anzahl der CPU's an „`dplace`“. Dies findet, an dieser Stelle, anhand einer Indexierung, beginnend bei Null, statt. Im Fall des Beispielskriptes werden, der Queue-Werte und der Threadanzahl angepasst, vier CPU's angegeben. Weitere Optionen können auf dem Orion in den Man-Pages mit „`man dplace`“ nachgelesen werden. Sind keine expliziten Angaben im „`dplace`“ Kommando enthalten, werden die voreingestellten Standardwerte der jeweilig gewählten Queue verwendet. Für die Programmentwicklung kann in die letzte Zeile des Beispielskripts vor „`dplace`“ die Anweisung „`time histx`“ eingesetzt werden, um z.B. Programmlaufzeiten nachvollziehen zu können. Dabei ist zu beachten, dass bei der Ausführung die Parallelisierungsmöglichkeiten nur eingeschränkt genutzt werden.

3.2.2. Programmparallelisierung mit OpenMP und Fortran90

OpenMP ist die Abkürzung für „**Open** specifications for **M**ulti **P**rocessing“ [15]. Sie ist eine Programmierschnittstelle für Fortran und C/C++, die wie schon beschrieben, zum Zweck der parallelen Programmierung mittels des Shared-Memory-Ansatzes für Mehrprozessor-Systeme entwickelt wurde. Durch die Anwendung von OpenMP Compiler-Direktiven sowie spezieller Unterprogramme wird die Abarbeitung bestimmter Programmteile auf mehrere Threads aufgeteilt. Dabei ist der Master Thread (Monitor Thread) mit der Nummer 0 in einem OpenMP-Programm immer aktiv. Der Programmierer bestimmt mit den OpenMP-Direktiven im Quellcode, wann sich das Programm in mehrere Threads aufteilt bzw. die Threads wieder zu einem vereint werden.

In die Parallelisierung mit OpenMP, wird folgend anhand eines praktischen Beispielprogramms in Fortran90 eingeführt. Dazu wird zunächst die Funktion des Bei-

spielprogramms „`Parallel`“ im Quellcode 3.3 auf der folgenden Seite erklärt. Das Programm verarbeitet ein großes Feld, das viel Hauptspeicher benötigt. Seine Werte werden in verschachtelten „`do`“-Schleifen jeweils mit individuell erzeugten Werten verrechnet. Das Feld „`bsparr`“ wird in Zeile `4` deklariert. Es besitzt 400 Mio. Werte vom Typ „`REAL*8`“ und die Dimensionen 20000 × 20000. In Zeile `8` wird das Feld mit Zufallswerten gefüllt. Die Zeilen `17 - 29` zeigen die Verarbeitung des Feldes in zwei geschachtelten „`do`“-Schleifen. Dieser Programmteil soll parallelisiert werden.

Um OpenMP-Direktiven im Programm nutzen zu können muss dem Programm im Deklarationsteil die OpenMP-Bibliothek mit „`use omp_lib`“ bekannt gemacht werden. Wenn, wie in Zeile `14` hingewiesen wird, im Queue-Skript OpenMP nicht mit der Anweisung „`OMP_NUM_THREADS = 4`“ die Anzahl der Threads bekannt gemacht wird, kann das, wie in Zeile `15`, mit dem Aufruf „`call omp_set_num_threads (4)`“ direkt im Quellcode realisiert werden. Da dies aber schon im Beispiel-Queue-Skript geschehen ist (s. Quellcode 3.2), ist diese Zeile auskommentiert. OpenMP-Direktiven sind als Kommentare gekapselt und beginnen in Fortran immer mit dem Ausdruck „`!$omp`“, dem sog. Wächter (engl.: *sentinel*). Darauf folgt eine OpenMP-Anweisung.

Im Quellcode 3.3 Zeile `18` ist die Anweisung „`parallel`“. Sie drückt aus, dass sich das Programm an dieser Stelle in die definierte Anzahl Threads aufteilen soll. Das Gegenstück, die Anweisung „`end parallel`“, in Zeile `28`, weist OpenMP an, die Threads an dieser Stelle des Programms wieder zu einem Thread zu vereinen. An die OpenMP-Anweisungen können Optionen angehängt werden, die die Abarbeitung des parallelisierten Programmabschnitts maßgeblich beeinflussen. Bei der `parallel`-Direktive kann z.B. die Sichtbarkeit, bzw. die Gültigkeit von Daten beeinflusst werden. Aufgrund des Shared-Memory-Ansatzes kann davon ausgegangen werden, dass Daten, mit Ausnahme von Schleifeniterationen, in OpenMP standardmäßig zwischen den Threads geteilt werden. Das ist nicht immer sinnvoll, z.B. wenn parallele Programmabschnitte Daten verarbeiten, deren Änderung auf ihre Nebenläufer einen negativen Einfluss haben. Dadurch kann es passieren, dass Threads ständig ein und den selben Speicherbereich verändern, sodass der Laufzeitvorteil durch die Parallelisierung ausbleibt. Die Gültigkeit bzw. Sichtbarkeit von Daten kann aus diesem Grund optional geändert werden. Mögliche Varianten für die `parallel`-Direktive sind z.B. „`shared`“, „`private`“ und „`firstprivate`“. Daten, die als „`shared`“ deklariert werden, sind explizit in allen Threads sichtbar und gültig. Die Änderung der Daten in einem Thread wirkt sich auf alle anderen aus. Daten, die als „`private`“

Listing 3.3: Beispiel: Parallelisierung mit OpenMP

```
      program parallel

      use omp_lib
      REAL*8, DIMENSION(20000,20000) :: bsparr
      REAL*8 dum1,dum2
      integer*8 i,j
      integer k
      CALL RANDOM_NUMBER(bsparr)
      dum1 = 1.5
      dum2 = 1.7
      i = 0
      j = 0
      k = 0
      !wenn im script kein OMP_NUM_THREADS=nr steht
!     call omp_set_num_threads(4)

        do i = 1, 20000
!$omp parallel private(j) shared(i,bsparr,dum1,dum2)
!$omp do
           do j = 1, 20000

          dum1 = dum1 + 0.000023
            dum2 = dum2 + 0.000072
            bsparr(j,i) =bsparr(j,i)*dum1/dum2

          end do
!$omp end do
!$omp end parallel
        end do

      end
```

deklariert werden, sind nur im aktuellen Thread sichtbar und gültig. Beim Eintritt in den parallelen Programmabschnitt werden sie nicht speziell initialisiert, weiter wirken sich die Änderungen dieser Daten nicht auf nachfolgende serielle Programmabschnitte aus. Jeder Thread besitzt also eine eigene Kopie einer Variablen in seinem Speicher, deren Wertigkeit nur er allein beeinflussen kann. Ein großer Vorteil wenn mehrere Threads unterschiedliche Teile einer Iteration gleichzeitig bearbeiten sollen. Der Unterschied von „`private`“ zu „`firstprivate`“ ist, dass die Daten im parallelen Programmabschnitt mit dem letztgültigen Wert aus dem vorhergehenden seriellen Programmabschnitt initialisiert werden. Mit der „`do`“-Direktive können `do`-Schleifen innerhalb eines `parallel`-Blocks parallelisiert werden, sodass die Schleifendurchläufe auf die einzelnen Threads bzw. CPU's aufgeteilt werden. Die Zuweisung der Schleifendurchläufe auf die Threads kann mit einer zusätzlichen `schedule`-Anweisung gesteuert werden. Ihr können wiederum unterschiedliche Argumente, mit Chunk[6]-Größen übergeben werden. Als Argumenttypen sind z.B. möglich „`static`“, „`dynamic`“, „`guided`“ oder „`runtime`“. Sie beziehen sich auf die Art der Threaderzeugung. Im Beispielprogramm könnte die „`do`“-Direktive auch so aussehen: „`!$omp do schedule(dynamic,4)`“. Darauf soll an dieser Stelle aber nicht weiter eingegangen werden.

Um ein Programm effizient zu parallelisieren genügt es nicht sich nur auf zusätzliche OpenMP-Direktiven im Quellcode zu beziehen. Auch aktive Code-Optimierung im Quellcode ist erforderlich um ein Programm möglichst effektiv mit kürzester Rechenzeit ablaufen zu lassen. Zum Einen sollten parallelisierte Programmabschnitte an die Verarbeitung mit OpenMP angepasst werden. Bei Programmlaufzeittests auf dem Orion-Server konnte z.B. durch die Zerlegung einer Formel mit intrinsischen Funktionen in mehrere Rechenabschnitte die Laufzeit nahezu halbiert werden. Die Berechnung von 400.000.000 Werten eines Feldes mit der unoptimierten Formel wie sie im Quellcode 3.4 zu sehen ist dauerte im Durchschnitt ca. 56 Sekunden. Durch die Zerlegung der Berechnungsformel in drei Teilergebnisse, deren Ergebnisvariablen jeweils Threadprivat sind und der Ausgliederung bzw. der Umwandlung, des Teils der Formel mit shared Variablen in eine Threadprivate Variable (s. Quellcode 3.5), konnte der durchschnittliche Programmdurchlauf auf ca. 26 Sekunden gesenkt werden. Weiter kann das Verarbeiten von großen Feldern optimiert werden, indem die

[6] zu Deutsch: *Stück*

Listing 3.4: Beispiel: Parallelisierung mit unoptimierter Berechnungsformel

```
      do j = 1, 20000
!$omp parallel private(i)
!$omp&shared(j,dumarr,dum1,dum2)
!$omp do
        do i = 1, 20000
            dum1 = dum1 + (0.000023,0.000044)
            dum2 = dum2 + (0.000072,0.000012)
            dumarr(i,j) = dumarr(i,j) + SIN(EXP(dum1 / dum2))
        end do
!$omp end do
!$omp end parallel
      end do
```

Listing 3.5: Beispiel: Parallelisierung mit OpenMP und optimierter Berechnungsformel

```
      do j = 1, 20000
!$omp parallel private(i,result1,result2,result3)
!$omp&shared(j,dumarr,dum1,dum2)
!$omp do
        do i = 1, 20000
            dum1 = dum1 + (0.000023,0.000044)
            dum2 = dum2 + (0.000072,0.000012)
            result1 = dum1 / dum2
            result2 = EXP(result1)
            result3 = SIN(result2)
            dumarr(i,j) = dumarr(i,j) + result3
        end do
!$omp end do
!$omp end parallel
      end do
```

Iterationsschleifen, die die Indexierung der einzelnen Feldwerte darstellen, der Art und Weise des Speicherauslesens angepasst werden [16]. In Fortran werden Felder spaltenweise im Speicher abgelegt. Das bedeutet für ein $m \times n$ Feld `a`, das im Speicher (speziell Cache) hinter Element `a(1,1)`, Element `a(2,1)` liegt und dahinter wiederum `a(3,1)`, usw.. Zu einem vom Prozessor angeforderten Element werden die ihm „nahen“ Elemente vom Hauptspeicher in den Cache geschrieben. Wenn die Feldelemente richtig, wie im Quellcode 3.3, der Reihe nach wie sie im Speicher vorliegen, ausgelesen werden, gibt es so wenig Lese- Schreibzugriffe zwischen Cache und Hauptspeicher wie möglich. Würden in diesem Fall die Feldelemente zeilenweise ausgelesen, wird je nach Feldgröße der Cache bei jedem vom Prozessor angeforderten Element gelöscht und neu beschrieben. Bei großen Feldern kann das zur Folge haben, dass sich die Programmlaufzeit deutlich erhöht. Wenn der selbe Programmablauf mit c/c++ wie im Beispielfortranprogramm realisiert werden sollte, müsste das Feld zeilenweise ausgelesen werden, um dem Speicherabbild des Feldes zu folgen. Diese und viele weitere Themen und Tipps zur Programmparallelisierung mit OpenMP sind in den Werken [10], [16] und [17] zu finden.

3.2.3. Abteilungsserver

Die Abteilungsserver sind für die Programmierung in MATLAB, Grads oder IDL vorgesehen, sowie der Nutzung von multimedialen Anwendungen, die hardwarebedingt auf dem Orion-Server nicht oder nur eingeschränkt lauffähig sind. Auf ihnen kann auch interaktiv gearbeitet werden wenn die Software Exceed genutzt wird. Durch sie kann die grafische Benutzeroberfläche Konquerror angezeigt werden, wenn im Exceed-Kommandofenster der Befehl „`konquerror`“ eingegeben wird. So können Programmierung, Textverarbeitung oder Grafikdarstellungen interaktiv auf dem Suse Linux 10 Betriebssystem des Abteilungsservers geschehen. Im Home-Verzeichnis jedes Benutzers ist auch das zum Nutzer gehörende Orion Home-Verzeichnis gemountet. So kann von den Abteilungsservern auch mit interaktiven Anwendungen auf dem Orion-Server gearbeitet werden ohne diese direkt auf dem Orion auszuführen. Das grafische Fenster von Grads beispielsweise, kann mit der Eingabe von „`gradsc`“ aufgerufen werden. Das Grads-Fenster für die `netcdf` Erweiterungen mit „`gradsnc`“. Die interaktive Programmierumgebung von IDL wird mit „`idlde`“ aufgerufen. Sie

Listing 3.6: Beispiel: IDL-Startskript für einen Abteilungsserver

```
1 #!/bin/csh
2
3 source /usr/local/rsi/idl/bin/idl_setup
4 setenv IDL_CPU_TPOOL_NTHREADS 1
5 setenv IDL_STARTUP IDLbatchfile oder Programmname
6 idl
7 unsetenv IDL_STARTUP
8 exit
```

stellt die selben Funktionen wie unter Windows zur Verfügung. Ein Programm, das eigenständig und unabhängig von der Benutzerkonsole auf einem Abteilungsserver laufen soll, kann in einem Batchskript gestartet werden. Dazu müssen bei IDL z.B. einige Parameter für die Laufzeitumgebung gesetzt werden, wie im Quellcode 3.6 zu sehen ist. Dort wird in Zeile drei das Startverzeichnis der IDL Laufzeitumgebung bekannt gemacht. In Zeile vier wird IDL explizit die Anzahl der Prozessoren übergeben die es während des Programmlaufs nutzen darf. Da IDL die vollen Ressourcen eines Rechners ausschöpft, muss diese Zeile auf den Abteilungsservern immer angegeben werden. Die maximal nutzbare CPU-Anzahl auf den Abteilungsservern beläuft sich auf zwei. Die darauf folgende Zeile übergibt den Namen der Batchdatei, bzw. den Programmnamen, das beim Start der IDL-Laufzeitumgebung ausgeführt werden soll. In Zeile sechs wird IDL gestartet und nach beenden des Programms in Zeile sieben der Startvorgang von IDL wieder auf den Standard zurückgesetzt.

3.2.4. Benchmark-Tests

Es wurden Testprogramme, sog. Benchmarks für die Abteilungsserver und den Orion geschrieben. Sie sind in Fortran90 implementiert und auf spezielle Rahmenbedingungen ausgerichtet, um ihr Laufzeitverhalten auf den verschiedenen Rechnern, sowie bei unterschiedlichem Parallelisierungsgrad betrachten und auswerten zu können. Die Rahmenbedingungen der Tests sind in Tabelle 3.2 veranschaulicht. Der Test

F zielt nicht auf die Laufzeitunterschiede bei unterschiedlichen Parallelisierungsgraden ab, sondern auf Zeitunterschiede beim Schreibprozess auf den DMF, zwischen den Abteilungs-Servern und dem Orion-Server. Bei den berechneten Aufgaben der Benchmarks wurde darauf geachtet, dass sich die Berechnungswerte bei einzelnen Schleifendurchläufen ständig ändern. Die Kompiler bzw. OpenMP sind so intelligent, das sie Quellcodesequenzen, die immer die selbe Berechnung mit immer den selben Werten erkennen und sie beispielsweise nur ein Mal ausführen. Ein Feld, das mit den Berechnungsergebnissen gefüllt werden soll, wird dann einfach komplett mit diesem einen Ergebnis beschrieben. Für die Tests **A** und **D** (s. Tab 3.2), wurde ein Algorithmus entwickelt, der im Quellcode 3.7 zu sehen ist. Ihm liegt die Berechnung von Kreiskoordinaten in einem kartesischen Koordinatensystem zu Grunde. Die Berechnung wiederholt sich in drei Schleifen. Die beiden inneren Schleifen, Zeilen `9` bis `19` stellen den eigentlichen Benchmarkalgorithmus dar. In ihnen werden 400 Mio.

Test	Feldgrößen (Speicher)	realer Speicher	Operationen
A (viel rechnen, wenig Speicher)	keine	1,872 MB	Gleitkomma Real*8 s. QBsp. 3.7
B (wenig rechnen, viel Speicher)	400 Mio. Werte (ca. 3 GB)	3 GB	Gleitkomma Real*8
C (mäßig rechnen, viel Speicher)	400 Mio. Werte (ca. 3 GB)	3 GB	Gleitkomma Real*8 s. QBsp. 3.5
D (viel rechnen, viel Speicher)	400 Mio. Werte (ca. 3 GB)	3 GB	Gleitkomma Real*8 wie QBsp. 3.7
E (mäßig rechnen, viel Speicher)	400 Mio. Werte (ca. 11,92 GB)	4,5 GB	Komplexe Werte Comp*32 wie QBsp. 3.5
F (viele DMF- Zugriffe)	2 Mio. / 2000 Werte (ca. 15,3 MB / 15,6 kB)	2,48 MB	Gleitkomma Real*8

Tabelle 3.2.: Spezifikationen der Tests. Der reale Speicherbedarf wurde unter „`top`“ , im unparallelisierten Durchlauf ermittelt. Die Datentypen der Felder entsprechen den Datentypen der Operationen.

Listing 3.7: Benchmark-Auszug: Parallelisierter Berechnungsalgorithmus

```
1        do k = 1,50
2 
3        t0 = secnds(0.0)
4  !$omp parallel private(i,j,r1,r2,r3,phi,x,y)
5  !$omp&reduction(+ : coord)
6  !$omp&firstprivate(k)
7  !$omp do
8 
9        do j = 1, 20000
10           do i = 1, 20000
11             r1 = ((REAL(i)/1000.0))
12             r2 = (sin(REAL(i) * (REAL(j)+REAL(k))))
13             r3 = r1 + r2
14             phi = (360.0 / 20000.0) * REAL(i)
15             x = r3*SIN(phi)
16             y = r3*COS(phi)
17             coord = coord +((x + y)/1000.0)
18           end do
19       end do
20 
21 !$omp end do
22 !$omp end parallel
```

Punkte errechnet. In der Berechnungsformel der Kreiskoordinaten ergibt sich der Wert für den Radius aus der Verrechnung der aktuellen Werte aller Iterationsvariablen „`i`“, „`j`“ und „`k`“. Dabei wird „`i`“ in Zeile `11` für die Vergrößerung des Kreisradiuses verwendet um eine Spirale zu erzeugen (s. Abb 3.3 (b)). In der darauf folgenden Zeile `12` wird unter Einbeziehung der aktuellen Iterationswerte ein Sinus-Wert erzeugt, der in Zeile `13` auf die Radiuserweiterung addiert wird. Dadurch entsteht auf der Spirallinie aus Abbildung 3.3 (b) ein zusätzlicher Sinusverlauf (s. Abb 3.3 (a)). Der Algorithmus ist zum Beispiel durch Vergrößerung einer der beiden inneren Schleifen beliebig erweiterbar. Um den korrekten Ablauf der Benchmarks nachvollziehen zu können werden alle Koordinaten in der Variablen „`coord`“ aufsummiert, sodass sie mit Ergebnissen aus Referenzprogrammen verglichen werden kann. Die

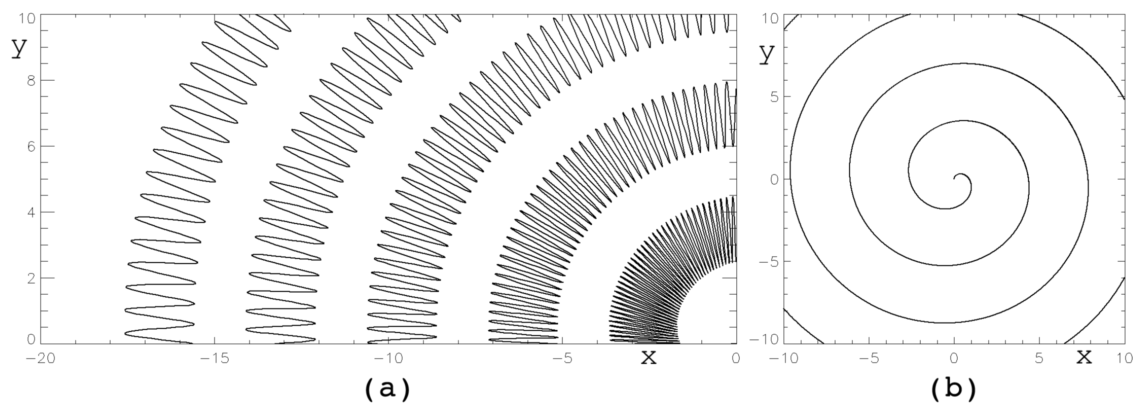

Abbildung 3.3.: Die Abbildung (a) zeigt eine Visualisierung der Ergebnisse des Benchmarkalgorithmus für den Benchmark *wenig Speicheroperationen, viele Rechenoperationen mit Gleitkommawerten.* Das Grundprinzip beruht auf der Berechnung von Koordinaten einer spindelförmigen Linie in einem kartesischen Koordinatensystem, zu sehen in (b).

Referenzprogramme sind in IDL implementiert und wurden auf einem Desktoprechner ausgewertet. Die äußerste Schleife ist für die mehrfache Wiederholung des eigentlichen Benchmarks implementiert worden, um bei Laufzeitunterschieden der einzelnen Rechendurchläufe Mittelwerte berechnen zu können. Zur Zeiterfassung wird die Funktion „`secnds()`“ wie in Zeile `3` verwendet. Der Genauigkeit dieser Funktion darf, laut der MAN-Pages mind. bis zu 1/10 s vertraut werden. Die Testprogramme, die *„mäßig“* viele Rechenoperationen ausführen, basieren auf dem Berechnungsalgorithmus, der in Quellcodebeispiel 3.5 veranschaulicht ist. Vor der Berechnungsschleife wird das Feld **„dumarr"** mit Zufallswerten gefüllt um eine dynamische Komponente in die Rechenoperationen einzubringen. *„Mäßig rechnen“* bezieht sich bei den Benchmark-Tests **C** und **E** auf den Algorithmus, der bei Test **E** durch das „`Complex32`“-Format der Daten aber zu sehr vielen Rechenoperationen führt. Die Benchmarkprogramme, die viele Speicheroperationen ausführen, arbeiten zusätzlich mit Feldern, die 400 Mio. Werte besitzen. Da die Laufzeitunterschiede zwischen den verschiedenen Benchmarkprogrammen deutlich variierten, sind die Ergebnisse in Abbildung 3.4 in normierter Form dargestellt. Bei allen Benchmarks der Abbildung 3.4 ist zu erkennen, dass ihre Parallelisierung mit zwei CPU's die Programmlaufzeit um ca. 50 % senkt. Die Benchmarkrechnung **B** mit wenig Floating Point Operationen im Verhältnis zu vielen

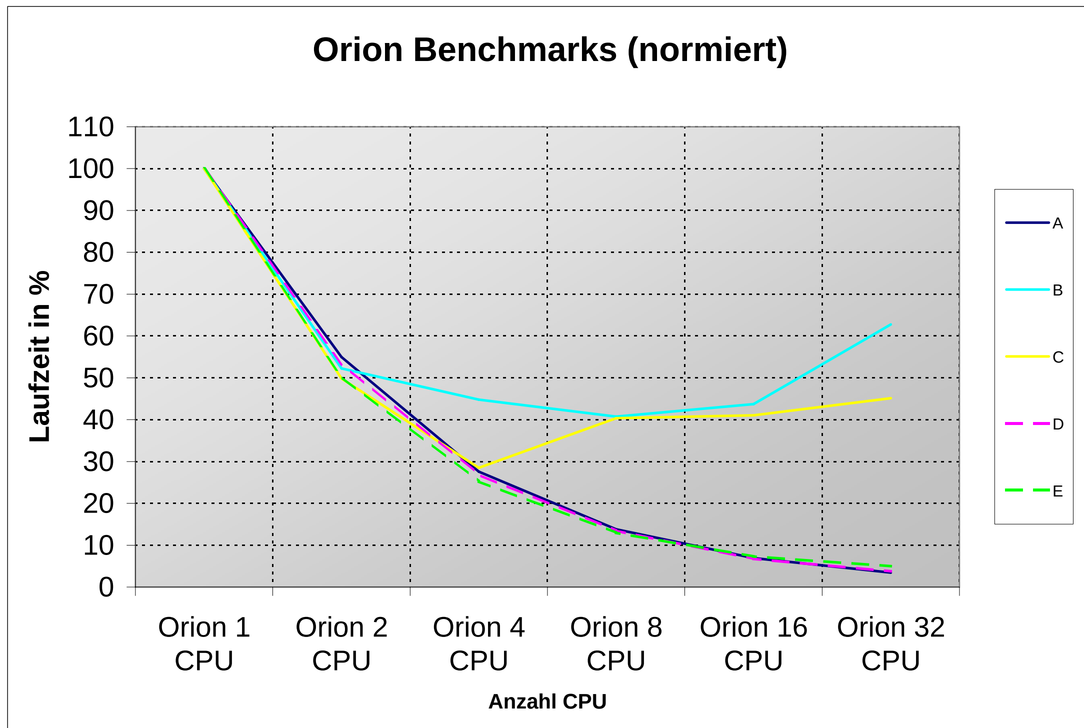

Abbildung 3.4.: Normierte Darstellung der Benchmarkergebnisse für die Programmparallelisierung auf dem Orion-Server.

Speicheroperationen hat gezeigt, dass eine Parallelisierung über zwei Prozessoren hinaus nur einen geringen bzw. keinen Laufzeitvorteil erzielt. Fast das selbe Ergebnis erzielte der Benchmark **C** mit mäßig vielen Gleitkommaoperationen und vielen Speicheroperationen. Hier lohnt noch die Parallelisierung mit vier Prozessoren. Alle Versuche über vier CPU's hinaus ergaben ausschließlich längere Laufzeiten. In dem Diagramm ist zu sehen, dass sich die Laufzeiten der beiden Benchmarks bei einer Parallelisierung mit acht Prozessoren auf einen Punkt annähern. Das liegt sehr wahrscheinlich ab diesem Parallelisierungszustand daran, dass die Rechenaufgabe nicht mehr lokal auf den Prozessoren eines *Blades* verteilt wird, sondern auf zwei *Blades*, womit sich auch die Kommunikationswege und -zeiten verlängern. Die Kurven der Benchmarks **A**, **D** und **E** folgen den idealen Erwartungen einer Parallelisierung. Bei verdoppelter CPU-Anzahl wird die Programmlaufzeit halbiert. Im Fall des Bench-

marks **A** wird die Rechenlast auf die jeweils verfügbaren CPU's aufgeteilt, ohne dass sie häufig auf einen gemeinsamen Speicher zugreifen müssen. Beim Benchmark **E** resultiert die ideale Parallelisierung aus dem Verhältnis des relativ geringen Kommunikationsaufwandes zum hohen Rechenaufwand. Dieser Benchmark arbeitet mit den selben Feldgrößen wie die anderen Benchmarks mit vielen Speicheroperationen, aber mit 256 Bit großen „`Complex32`"-Werten, gegenüber 64 Bit großen „`Real*8`"-Werten. Obwohl sie den selben Rechenalgorithmus verwenden, betrug die Rechenzeit des Complex32-Benchmarks **E** im unparallelisierten Zustand ca. das 24-Fache der Laufzeit des Benchmarks **C** mit Gleitkommawerten. D.h. trotz des großen Speicherbedarfs werden die Kommunikationszeiten im Benchmark-Test **E** zum größten Teil durch den hohen Rechenaufwand überlagert. Die Tests **A** und **D** besitzen ebenfalls jeweils den selben Rechenalgorithmus. Gegenüber Test **A** wird in Test **D** zusätzlich ein 400 Mio. Feld beschrieben. Der direkte Vergleich der Programmlaufzeiten in Tabelle 3.3 ergibt, dass Test **D** für die Programmdurchläufe zwischen 5% und 14% mehr Zeit benötigt als Test **A**. In diesem Fall ist beim Test **D** das Verhältnis von Rechenaufwand zu den Speicheroperationen noch so ausgewogen, dass es auf die Effektivität der Parallelisierung kaum Auswirkungen hat (s. Abb. 3.4).

Die Parallelisierungs-Benchmarks auf dem Orion-Server zeigen, dass der Grad einer Programmparallelisierung nach dem Verhältnis von Rechenaufwand zu Speicheroperationen gewählt werden sollte. Wird der Parallelisierungsgrad unangemessen hoch gewählt, kann sich die Programmlaufzeit gegenüber dem Idealparallelisierungspunkt erhöhen, außerdem wird ein Großteil der verwendeten Hardwareressourcen sinnlos blockiert und belastet.

Test	**A**	**B**	**C**	**D**	**E**
Photon 1 CPU	1069,07	240,28	888,78	1105,26	6459,12
Orion 1 CPU	1972,12	117,43	214,26	2096,47	5218,21
Orion 2 CPU	1083,82	61,37	106,80	1120,26	2621,91
Orion 4 CPU	544,21	52,57	61,07	563,00	1318,07
Orion 8 CPU	272,62	47,82	86,46	282,39	676,05
Orion 16 CPU	136,43	51,35	87,93	141,44	380,75
Orion 32 CPU	68,25	73,70	96,71	78,49	258,92

Tabelle 3.3.: Laufzeiten der Benchmarktests in Sekunden.

Die Laufzeitverhalten dieser fünf Benchmarks sind auch auf dem Abteilungsserver Photon, in der unparallelisierten Form getestet worden, um die Rechenperformance der Abteilungsserver mit dem Orion-Server vergleichen zu können. In der Abbildung 3.5 sind diese Ergebnisse in normierter Form dargestellt. Dabei ist das Benchmarkergebnis mit der längeren Laufzeit als 100 % festgelegt, gegenüber dem dann das kürzere Laufzeitergebnis in % abgelesen werden kann. Der erste Vergleich links in der Abb. 3.5 zeigt, dass der Photon-Server ca. 45 % weniger Zeit für den Benchmark **A** mit *vielen Floating Point Operationen und wenig Speicheroperationen* benötigt. Da bei diesem Benchmark die Rechenleistung der CPU's fokussiert wird, ist der Abteilungsserver mit seinem deutlich höherem CPU-Takt (s. Kap 2.1) gegenüber der Orion-CPU im Vorteil. Bei den folgenden zwei Benchmarks **B** und **C**

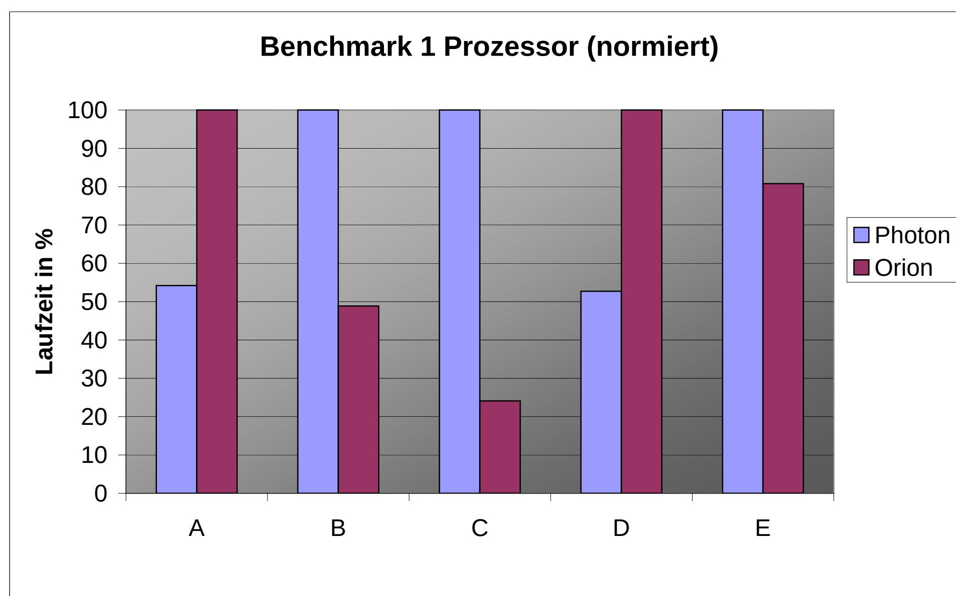

Abbildung 3.5.: Normierte Darstellung der Benchmarkergebnisse mit einem Prozessor für den direkten Vergleich des Orion-Servers und des Abteilungsservers Photon.

mit viel Speichernutzung ist das Gegenteil der Fall. Hier ist der Orion mit seinen großen Hauptspeicherressourcen, gegenüber den Abteilungsservern viel besser geeignet. Beim Benchmark-Test **D** ist auf dem Photon das Selbe zu beobachten wie beim Vergleich der Parallelisierung auf dem Orion zwischen Test **A** und **D**. Der Rechenaufwand überwiegt gegenüber den Speicheroperationen so sehr, dass sich die Laufzeit von Test **D** gegen über **A** nur geringfügig ändert. Der letzte Vergleich für den Benchmark **E** mit *mäßig Complex32 Operationen und vielen Speicheroperationen* rechts in der Abbildung zeigt, dass der Photon-Server trotz des hohen Speicherbedarfs nur ca. 20 % mehr Zeit benötigt als der Orion. In diesem Fall spielt wieder die Taktfrequenz der unterschiedlichen Prozessoren aber auch das Verhältnis von Rechenoperationen zu Speicheroperationen eine Rolle, die den Unterschied der Laufzeitergebnisse verringern.

Der Test **F** für die Zugriffe auf den DMF ist in zwei verschiedenen Versionen ausgeführt worden. In der ersten Version werden 50 Dateien auf dem DMF angelegt, in die je ein Feld mit 2 Mio. Werten geschrieben werden. Die Datei besitzt eine Größe von etwa 15,3 MB. Unterschiede zwischen dem Abteilungsserver Photon und dem Orion-Server konnten nicht beobachtet werden. Die Laufzeiten der Programme beliefen sich je nach Auslastung des DMF's zwischen 12 und 40 Sekunden. In der zweiten Version werden 500 Dateien auf dem DMF angelegt, in die ein Feld mit 2000 Werten geschrieben wird. Die Größe der Dateien beträgt ca. 15,6 kB. Auch hier konnten keine signifikanten Unterschiede festgestellt werden. Dabei benötigte der Photon-Server im Schnitt ca. 4 Sekunden und der Orion ca. 2 Sekunden für das Anlegen aller Dateien.

4. Strömungsfilm

Im folgenden Kapitel werden die theoretischen Untersuchungen und die Realisierung des Strömungsfilmes beschrieben.

4.1. Motivation

Im folgenden Abschnitt wird kurz erläutert mit welchen Mitteln die Abteilung *Optische Sondierungen* die Aufgabenfelder des IAP bearbeitet. Anschließend wird auf das Leibniz-Institut Middle Atmosphere (LIMA)-Modell eingegangen aus dessen Ergebnisdaten die Erzeugung eines Strömungsfilms angestrebt wird.

4.1.1. Die Abteilung Optische Sondierungen

Am IAP-Kühlungsborn werden, wie in der Einleitung schon erwähnt, verschiedenste Prozesse in der mittleren Atmosphäre untersucht. In der wissenschaftlichen Abteilung *Optische Sondierungen* geschieht dies anhand von Lidarsystemen, Ballonmessungen, hochauflösenden digitalen Kameras und verschiedenen Computermodellen.

Die unterschiedlichen Lidarsysteme basieren alle auf demselben Funktionsprinzip. Sie bestehen aus einer Sendeeinheit und einer Empfangseinheit. Die Sendeeinheit besteht aus einem leistungsstarken Laser, der Lichtimpulse in die Atmosphäre aussendet. Das von den Luftmolekülen zurückgestreute Licht wird von einem Teleskop fokussiert und in einen Nachweiszweig geleitet, wo das Licht dann nach gewünschten Wellenlängen selektiert und die Photonen anschließend von empfindlichen Detektoren gezählt werden. Diese Daten werden für jeden Lichtimpuls von einem Mess-PC

für eine Auswertung gespeichert. Durch die unterschiedlichen Laufzeiten der Photonen ergibt sich für jeden Laserpuls ein Höhenprofil des gestreuten Signals. Aus den Lidarmessungen werden Temperatur- und Winddaten gewonnen, sowie daraus resultierende Informationen über Wellenaktivitäten in der Atmosphäre [18].

Mit den Ballons werden zeitlich sehr hoch aufgelöste Wind- bzw. Turbulenzmessungen in der Atmosphäre vorgenommen. Die Messinstrumente, die als angehängte Last vom Ballon mehr oder weniger vertikal durch die Atmosphäre transportiert werden, sind Radiosonden und sog. Hitzdrahtanemometer [19].

Durch die gewählte Anordnung der Beobachtungskameras in Europa (s. Abb. 1.1 Seite 3) kann die räumliche Ausdehnung sowie die horizontale Struktur von leuchtenden Nachtwolken auf kontinentalen sowie kleinen Skalen beobachtet werden. Mit diesem vollautomatisierten Kameranetzwerk wird insbesondere die Häufigkeit der südlichen Ausdehnung, der NLC bis in niedrige Breiten untersucht.

Die Modellierung und theoretische Interpretation der experimentell gemessenen Atmosphärenvorgänge, vorrangig im Bereich der Mesosphäre und unteren Thermosphäre, ist ein weiterer wichtiger Bestandteil der Arbeiten dieser Abteilung. In der vorliegenden Arbeit wird speziell mit den Ergebnisdaten des LIMA-Modells gearbeitet, die verschiedenste Visualisierungen vorraussetzen um Informationen aus ihnen gewinnen zu können.

4.1.2. Das LIMA-Modell

Im Jahr 2005 wurde eine Version des Zirkulationsmodells LIMA am IAP fertiggestellt mit der klimatologische Studien durchgeführt werden. Es beschreibt die wichtigsten Prozesse der mittleren Atmosphäre, bestehend aus Dynamik, Strahlung, Chemie und Transport. Das Modell nutzt eine global hochaufgelöste sphärische Dreiecksgitterstruktur und den Einsatz von Datenassimilationstechniken für eine reale Beschreibung der Troposphäre und unteren Stratosphäre. LIMA ist das erste Global Circulation Model (GCM), das die mittlere Atmosphäre auf der Basis einer Dreiecksgitterstruktur beschreibt. Das aktuell vom LIMA-Modell genutzte Dreiecksgitter besitzt eine Kantenlänge von ungefähr 110 km und 41804 Gitterpunkte. Abbildung 4.1 zeigt eine Visualisierung des Gitters mit 270 km Kantenlänge und 6812 Gitterpunkten. Der wesentliche Vorteil des Dreiecksgitters gegenüber dem

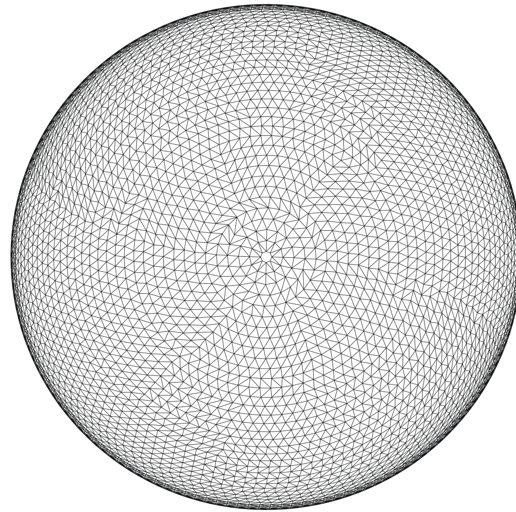

Abbildung 4.1.: LIMA Dreiecksgitterversion mit 270 km Kantenlänge

klassischen Längen- Breitengradgitter ist, dass die räumliche Auflösung überall nahezu konstant ist. Dies ist für die Beschreibung von kleinskaligen Prozessen im Modell, wie z.B. Wellen sehr wichtig. Vertikal ist das LIMA-Modell mit einem Gitterpunktabstand von ca. Δz=1.1km in 118 Schichten, auf jeweils gleichem Druckniveau aufgelöst.

Die vom Modell genutzten Daten der Tropo- und Stratosphäre werden aus täglich global gesammelten atmosphärischen Messungen vom European Center for Medium range Weather Forecast (ECMWF) zur Verfügung gestellt. Das ECMWF Datenarchiv beinhaltet meteorologische Daten in einem Zeitraum von 1960, bis zum aktuellen Zeitpunkt. Die einzelnen ECMWF-Datensätze liegen in einem globalen, horizontalen Spektralraster von $1°\times 1°$ auf 21 vertikalen Druckniveaus (0-45 km Höhe), für jeweils alle sechs Stunden eines Tages vor. Die Enbindung der ECMWF-Daten in das LIMA-Modell erfolgt, indem sie auf das horizontale Dreiecksgitter in einem Höhenbereich von 0-40 km interpoliert werden (s. Abb. 4.2). Parallel zu allen in LIMA ablaufenden atmosphärischen Prozessen in 0-40 km Höhe werden die ECMWF-Größen Horizontalwind, Temperatur und Bodengeopotential mittels einem sog. *nudging*-Verfahren angepasst, wobei die ECMWF-Daten innerhalb eines sechs Stunden Zeitintervalls linear interpoliert werden.

Das Modell ist mit Fortran90 implementiert worden und wird aktuell auf dem

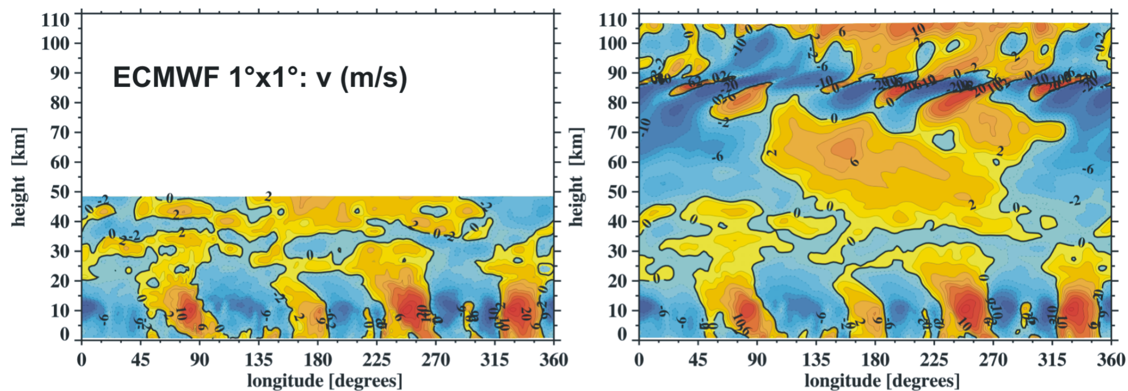

Abbildung 4.2.: Vertikalschnitte vom 10. Juli 2005 (00:00 UT) für die Breite Kühlungsborn (54N) des Meridionalwind (m/s) mit ECMWF Daten (links) und LIMA Modelldaten (rechts) [3].

Orion-Server ausgeführt. Das laufende Programm speichert Ergebnisdateien global in einer zeitlichen Auflösung von sechs Stunden, das bedeutet, dass für jeden simulierten Tag vier Dateien angelegt werden. Jede Datei ist 112 MB groß und wird binär im unformatierten Fortranformat abgelegt. Auf jedem Gitterpunkt des Modells werden die Größen Zonalwind [m/s], Meridionalwind [m/s], Temperatur [K], geometrische Höhe [m], Dichte [g/cm^3] und Vertikalwind für jeden Zeitschritt berechnet. Das bedeutet, dass jede dieser Größen in einem Feld der Dimensionen Gitterpunktanzahl (41804) $\times$ Anzahl der Höhenbereiche (118) vorliegen. Sie werden durch die simulierten physikalischen und chemischen Prozesse im Modell verändert.

Die meisten Messsysteme und Modelle dieser Abteilung werden eingesetzt um die Eigenschaften von NLC zu erforschen. Diese stellen wiederum einen Indikator verschiedener Eigenschaften der mittleren Atmosphäre dar. Das Verhalten der NLC kann nicht allein mit einer meteorologischen Größe, wie z.B. Temperatur erklärt werden. Um Zusammenhäge der Temperatur in Verbindung mit Luftströmungen untersuchen zu können, soll eine entsprechende Strömungsvisualisierung realisiert werden.

In einer vorausgegangen Diplomarbeit ([2]) ist eine Software mit IDL entstanden, die die einzelnen Größen des LIMA-Modells in polarstereografischen Plots visualisiert und anschließend zu einem Film zusammenfügt. In dieser Arbeit ist die Idee zur Erstellung eines Strömungsfilms anhand der LIMA-Ergebnisdaten kurz betrachtet und ein hoher Informationsgehalt dieser Visualisierungsart meteorologischer Daten festge-

stellt, aber nicht weiter bearbeitet worden. An dieser Stelle wird in der vorliegenden Arbeit die Idee des Strömungsfilms wieder aufgenommen und Untersuchungen zur Realisierung durchgeführt.

4.2. Visualisierung

In den folgenden Abschnitten wird in das Thema Visualisierung eingeführt. Es erfolgt eine Betrachtung der Grundlagen sowie eine Themeneinordnung der Aufgabenstellung.

4.2.1. Einführung

Bildhafte Darstellungen sind ein weit verbreitetes Medium zur Präsentation von Informationen, um Erkenntnisse intuitiv vermitteln zu können. Sie wurden schon lange vor dem Computerzeitalter, in der Kunst und Wissenschaft angewendet. Seit der technologischen Entwicklung der Rastergrafik-Hardware Anfang der siebziger Jahre des 20. Jahrhunderts startete die Computergrafik in den siebziger und achtziger Jahren ihren Siegeszug in der wissenschaftlichen und der High-end-Anwendungsdomäne. Seitdem ist es an fast jedem Rechnerarbeitsplatz üblich und vor allem möglich geworden Daten und Zusammenhänge grafisch darzustellen, um eine effiziente Analyse und Kommunikation zu ermöglichen. Im Mittelpunkt der aktuellen wissenschaftlichen Entwicklung der Computergrafik stehen die Anwendungen, allen voran der Einsatz von Computergrafiktechniken in speziellen Teildisziplinen, die die Visualisierung als eine von ihnen charakterisiert:

- Visualisierung, speziell wissenschaftliche Visualisierung und Informationsvisualisierung;
- Computeranimation;
- virtuelle und erweiterte Realität und allgemein virtuelle Umgebungen [20].

Grundlegend wird der Prozess der Erzeugung von Darstellungen aber als Visualisierung bezeichnet [21]. Ausgangspunkt jeder Visualisierung sind die darzustellenden

Informationen, aufgrund dessen die Visualisierung auch als Berechnungsmethode definiert wird, die symbolische Informationen in geometrische transformiert und so z.B. Wissenschaftlern die Betrachtung der Ergebnisse ihrer Simulationen und Berechnungen in Form von Bildern ermöglicht. Sie hat die Aufgabe *Ergebnisse zu präsentieren* und damit das Verständnis über die Daten und die zugrunde liegenden Konzepte zu erleichtern. Weiterhin soll sie die *Analyse der Daten* unterstützen, indem die Grafiken so aufgebaut sind, dass der Betrachter in der Lage ist Informationen aus den visualisierten Daten zu erkennen, zu verstehen und zu bewerten. Im Idealfall sollen innere verborgene Zusammenhänge aufgezeigt werden, die allein aus der Interpretation der Rohdaten nicht ableitbar wären. Die Visualisierung ist darum auch immer in einen kreativen Prozess eingebunden, der die Motivation besitzt Strukturen und Zusammenhänge aufzudecken und darüber zu kommunizieren. Um abstrakte Daten grafisch darstellen zu können, muss eine geometrische Beschreibungsform für eine Abbildung gefunden werden. Die Definition geeigneter Abbildungen, die eine Erfüllung der genannten Aufgaben einer Visualisierung gewährleisten, ist keine einfache Aufgabe. Bei der falschen Wahl der Abbildung können Grafiken entstehen, die zu Fehlinterpretationen der dargestellten Daten führen.

4.2.2. Anforderungen an Visualisierungen

Die vorliegende Arbeit ist in der Teildisziplin wissenschaftliche Visualisierung der Computergrafiktechniken einzuordnen. Die wissenschaftliche Visualisierung hat die Aufgabe, geeignete visuelle Repräsentationen einer gegeben Datenmenge zu erzeugen, um damit eine effektive Auswertung zu ermöglichen. Durch sie soll die Analyse, das Verständnis und die Kommunikation von Modellen, Konzepten und Daten in der Wissenschaft erleichtert werden. Sie wird in den drei Stufen *explorative Analyse*, *konfirmative Analyse* und *Präsentation* eingesetzt.

Die *explorative Analyse* wird angewendet, wenn noch keine Hypothesen über die vorhandenen Daten und ihre Eigenschaften existieren. Die Daten sind in dieser Phase selbst der Ausgangspunkt für die Darstellung. Oft wird eine ungerichtete Suche nach Informationen und Strukturen in den Daten durchgeführt. Es wird für die Visualisierung eine Darstellung gesucht, die Hinweise zur Formulierung einer Hypothese über die Daten und ihren Hintergrund gibt.

Wenn für die Daten eine Hypothese vorliegt, bildet das in Verbindung miteinander die Grundlage für eine *konfirmative Analyse*. Ihr Ziel ist es die vorhandene Hypothese mit einer geeigneten Visualisierung zu überprüfen.

Der letzte Schritt des Analyseprozesses stellt die *Präsentation und Kommunikation* der erzielten Ergebnisse dar. Zu diesem Zeitpunkt liegen bestätigte Aussagen und Fakten aufgrund der beiden vorangegangen Prozesse vor. Die Visualisierung muss an dieser Stelle die Fakten so erkennbar darstellen, dass dritte Betrachter sie ohne Probleme identifizieren und verstehen können.

Dies fand im Bezug auf die LIMA-Daten in einer vorausgegangenen Diplomarbeit ([2]) statt. In ihr sind die einzelnen Größen der Ergebnisdaten, in polarstereografischen Plots, für sich visualisiert worden. Diese Visualisierung ist zurückschauend betrachtet in alle drei Stufen einzuordnen. Sie wurde zur *explorativen Analyse* verwendet, die neue Strukturen in den Daten hervorbrachte, aber auch zur *konfirmativen Analyse* genutzt um die korrekte Arbeitsweise des Modells zu verifizieren. Weiterhin wurde sie zur *Präsentation* eingesetzt, weil bei der Visualisierung von Anfang an die Erfahrungen der Wissenschaftler einflossen und so eine optimale *Kommunikation* erreicht werden konnte. Die Visualisierung der vorliegenden Arbeit soll wieder den Anspruch aller drei Stufen erfüllen. Dabei soll besonders auf eine noch bessere *Kommunikation* durch die Kombination unterschiedlicher Größen der LIMA-Daten hingearbeitet werden.

Die Qualität einer Visualisierung hängt von vielen verschiedenen Einflussfaktoren ab, die im Vorhinein beachtet werden müssen. Die Bewertung einer Visualisierung hängt z.B. davon ab, ob die Art und die Struktur der Daten Beachtung finden. Welcher Typ von Daten sowie Dimension und Struktur des Betrachtungsbereichs werden dargestellt? Das Bearbeitungsziel der Visualisierung kann beispielsweise ein Überblick, eine Detailanalyse oder eine Ergebnispräsentationen für Dritte sein. Weiter ist auf die Eigenschaften des Zielpublikums einzugehen, auf das Vorwissen oder die visuellen Fähigkeiten und Vorlieben des Betrachters. Es sollten übliche Metaphern des Anwendungsgebietes wie z.B. übliche Symbole oder Darstellungsformen beachtet werden. Und auch die Charakteristika des Darstellungsmediums, wie Auflösung, Anzahl der darstellbaren Farben, Rechenleistung usw. sollten erörtert werden.

Die allgemeinen Ziele zur Erstellung einer guten Visualisierung, auf die primär hinaus gearbeitet werden sollte sind Expressivität, Effektivität und Angemessenheit. Die Expressivität oder Ausdrucksfähigkeit beschreibt die Unverfälschtheit der darge-

stellten Daten. Sie ist primär von der Struktur und der Art der Daten abhängig, die visualisiert werden sollen. Einzelne Visualisierungstechniken lassen sich z.B. bezüglich ihrer Expressivität für unterschiedliche Daten klassifizieren. Das allein ist jedoch nicht ausreichend, um sicher zu stellen, dass eine anschauliche visuelle Darstellung der gegebenen Daten generiert wird. Für eine Datenmenge kann es mehrere grafische Darstellungsarten geben, die das Expressivitätskriterium erfüllen. Die unterschiedlichen Darstellungsarten, die in Frage kommen, müssen nun danach untersucht werden welche von ihnen die Daten am besten präsentieren. Dabei muss z.B. auf die visuellen Fähigkeiten des Betrachters, sowie die charakteristischen Eigenschaften des Ausgabegerätes eingegangen werden. Das Effektivitätskriterium gibt also Aufschluss über die Fähigkeit einer Darstellungsart, die in ihr enthaltenen Informationen auf intuitive Weise dem Betrachter zu vermitteln. Die beiden vorangegangenen Anforderungskriterien umfassen alle Faktoren, um die effektivste Visualisierung einer gegebenen Aufgabenstellung zu erreichen. Die Visualisierung von Daten ist jedoch auch mit verschiedenen Aufwänden, wie z.B. Arbeits- und Kostenaufwand verbunden, sodass auch die Frage nach der Angemessenheit gestellt werden muss, in der die Aufwände dem Nutzen gegenüber gestellt werden. Die Angemessenheit beschreibt also weniger die Qualität der Visualisierung, sie umfasst den Rechen- und Ressourcenaufwand, der zur Generierung der visuellen Darstellung notwendig ist.

4.3. Theoretische Vorbetrachtungen

Für die LIMA-Daten wird eine Visualisierungsform angestrebt, in der Temperaturen und Wind, also Luftströmungen gleichzeitig dargestellt werden. Die verschiedensten meteorologischen Darstellungen von Winddaten wurden bereits in der vorausgegangenen Arbeit ([2]) betrachtet. Durch die Untersuchungen wurde eine Visualisierungsform fokussiert, die im Jahr 2004 von der ARD beim Wetterbericht eingeführt wurde und auch aktuell noch gezeigt wird. Entwickelt wurde die Strömungsvisualisierung (s. Abb. 4.3) an der Universität Basel, veröffentlicht von der Website www.Meteoblue.com. Seit 2006 ist Meteoblue ein eigenständiges Unternehmen, das seine meteorologischen Dienstleistungen aktuell zum größten Teil nur noch kommerziellen Nutzern zur Verfügung stellt, worunter auch die Strömungsvisualisierung fällt. Das Material, das zur Analyse der Strömungsvisualisierung von Meteoblue verwendet wird, wurde zur Zeit der ersten Untersuchungen 2006 von der Website frei zur Verfügung gestellt. Bis zum aktuellen Zeitpunkt gibt es auf dem Gebiet der meteorologischen Strömungsvisualisierungen keine bekannten Weiterentwicklungen, die eine

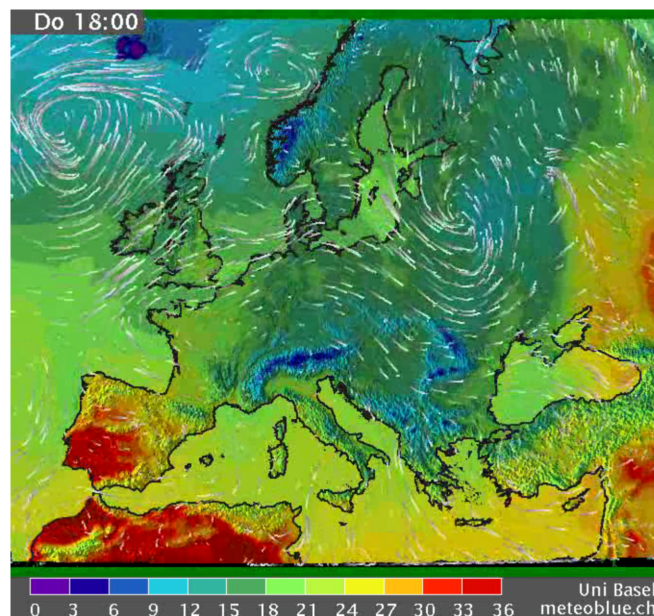

Abbildung 4.3.: Beispielbild aus einem Strömungsfilm der Universität Basel, der die Vorlage des gewünschten Strömungsfilms ist [22].

höhere Qualität als die der ursprünglichen Strömungsvisualisierung der Universität Basel aufweisen.

Die Strömungsvisualisierung wurde laut der Website in 2006 mittels IDL und Java-3D realisiert. Informationen, die der Visualisierung entnommen werden können, sind die farblich kodierten Temperaturen in Bodennähe, die zusätzlich auf eine dreidimensionale Reliefkarte gelegt sind. Die Landmassen sind durch eine deutliche schwarze Line vom Wasser getrennt, sodass eine einfache Orientierung des Betrachters möglich ist. Die atmosphärische Strömung der Luft in Bodennähe wird durch Strömungslinien dargestellt. Die Strömungslinien geben die Windrichtung und die relative Geschwindigkeit wieder. Standardmäßig werden in der Meteorologie Windpfeile verwendet die eine feste Koordinate besitzen. Aus diesem einen festen Punkt zeigt der Pfeil in die Windrichtung und gibt die relative Windstärke mit seiner Länge an. Bei einer Windrichtungsänderung dreht er sich um diesen Punkt. Eine genauere meteorologische Darstellung verwendet am Ende des Windpfeils, also an dem Teil der Linie, der in die Strömungsrichtung zeigt, keine Pfeilspitze sondern unterschiedliche sog. Fähnchen, die die Windgeschwindigkeit exakt erkennen lassen. Diese Darstellungsarten sind ursprünglich für Momentaufnahmen, also die genauere Betrachtung von Einzelbildern, für Windfelder entwickelt worden und sind dadurch für animierte Bildsequenzen nur mäßig- bzw. unbrauchbar. Eine andere, sehr aussagekräftige Methode bei der Visualisierung von Strömungsdaten ist die Liniendarstellung, wie sie auch in der Strömungsvisualisierung der Universität Basel verwendet wird. Diese Visualisierungsmethode bildet die Bahn einzelner Partikel, die sich in einem Strömungsfeld bewegen ab (s. Abb. 4.4 (a)). Genau genommen wird eine Verbindungslinie aller Orte, die das Partikel in einer Strömung im Laufe der Zeit erreicht, gezeichnet. Die Wegbahnen werden Trajektorien genannt. In der Abbildung 4.4 (a) besitzen die Strömungslinien als *Kopf* einen Pfeil um Anfang und Ende unterscheiden zu können. Der Pfeilkopf zeigt die letzte Position des Partikels an, dessen Weg beschrieben wird. An einigen Stellen der Abbildung sind, auf Grund ihrer Länge, mehrere Strömungslinien übereinander gezeichnet, sodass sie überladen wirkt. Außerdem geht in dieser Abbildung, durch die Unverhältnismäßigkeit von der Anzahl der Strömungslinien zu ihrer Länge, die Information über die Windgeschwindigkeit verloren. Der Vorteil der Strömungsvisualisierung der Universität Basel liegt in der Darstellung der Strömung. Genauer, in der Anzahl der Strömungslinien, ihrer Form und wie sie sich verhalten. Durch die ausgewogene Anzahl wird eine Überfüllung des Bildes mit

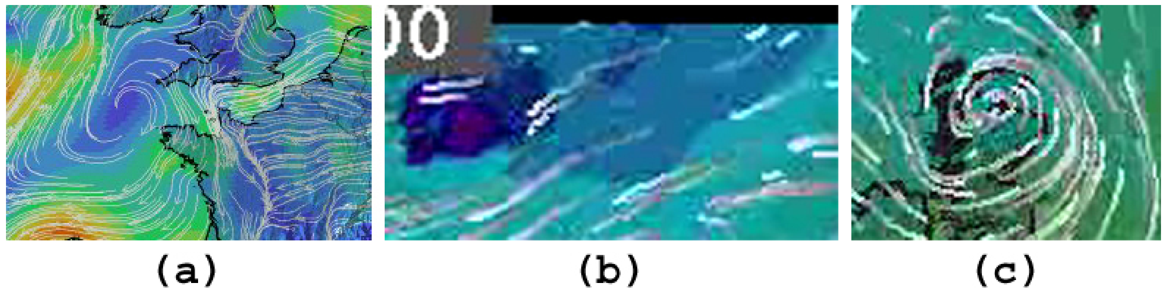

Abbildung 4.4.: Bild (a) zeigt Strömungspfeile in Kombination mit Trajektorien als Objekte für die Winddarstellung. Die Bilder (b) und (c) verdeutlichen die Eigenschaften Trajektorienverwendung, zeitlich beschränktes Weggedächtnis und Transparenz zum Ende der Linien, die die Luftströmung im angestrebten Beispielfilm darstellen [22].

Strömungslinien verhindert, aber dennoch die Bewegung der Luftmassen sehr gut verdeutlicht. Die Windrichtung und die relative Windgeschwindigkeit sind ebenfalls sehr gut aus einem Standbild (s. Abb. 4.3) ablesbar, ohne dass die Strömungslinien richtungweisende Pfeile besitzen. Das Problem des übermäßigen Überlagerns der Strömungslinien wird in der Visualisierung gelöst, indem erstens das Weggedächtnis der einzelnen Strömungslinien nur eine bestimmte kurze Zeit angezeigt wird und zweitens die Strömungslinie selbst eine begrenzte Lebensdauer besitzt. Nach Ablauf dieser Lebensdauer verschwindet die Strömungslinie an ihrem aktuellen Ort und wird erneut von ihrem festgelegten Ursprungsort aus in das Windfeld gezeichnet. Bei starker Vergrößerung (s. Abb. 4.4 (b) und (c)) wird deutlich, dass die Linien nach kurzer Strecke zum Ende hin die Farbe des Hintergrundes annehmen. Dies deutet auf einen Transparenzeffekt hin, der es ermöglicht Anfang und Ende der Linie eindeutig und schnell zu erfassen. Die Verdeutlichung der relativen Windgeschwindigkeit wird realisiert, indem das Weggedächtnis jeder Strömungslinie dieselbe Zeitdauer angezeigt wird, bevor sie verschwindet. Entsprechend wird eine geringe Windgeschwindigkeit nicht nur durch eine langsame Bewegung der Strömungslinie in der Animation, sondern auch durch ihre kurze Länge verdeutlicht.

4.3.1. Aktuelle Gegebenheiten und Voraussetzungen

Von den LIMA-Daten liegen visualisierte Einzelbilder aller enthaltenen Größen, für die Modellhöhenniveaus 2, 42 sowie 75, die der NLC-Höhe entspricht, vor. Sie sind in polarstereografischen Ansichten für beide Hemisphären erstellt worden, wie es in der Abbildung 4.5 für die Temperatur zu sehen ist. Zeitlich sind die Visualisierungen genau wie die LIMA-Daten in sechs Stunden aufgelöst. Die Visualisierungen liegen aktuell für einen Zeitraum von 1961 bis 2009 vor. Die Bildformate in denen die Plots vorliegen, sind Bitmapdateien vom Typ Portable Network Graphics (PNG). Sie besitzen eine Auflösung von 1024 Bildpunkten in der Breite und 768 Bildpunkten in der Höhe. Die Plots werden vollautomatisiert von einer Software generiert, die in IDL implementiert ist und mehr als 7000 Quellcodezeilen besitzt. Um einen ähnlichen Aufwand für eine erneute Visualisierung der Temperaturdaten zu vermeiden

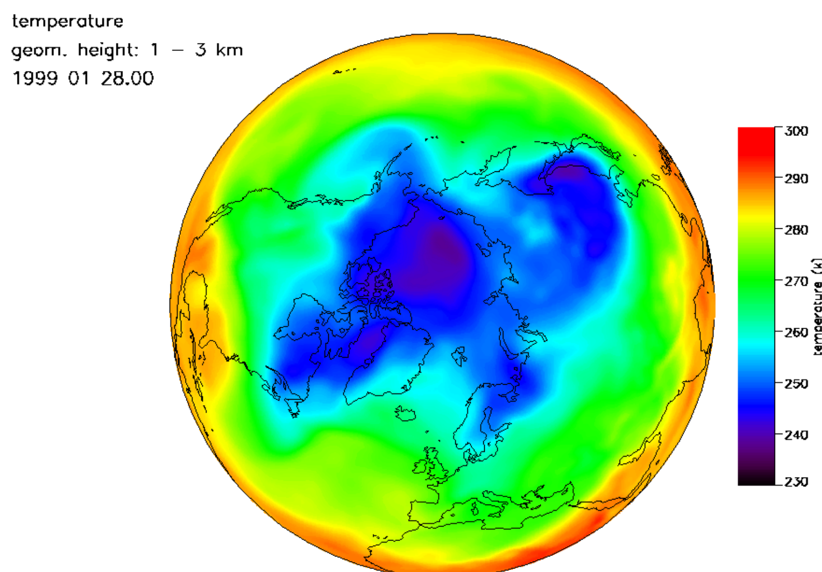

Abbildung 4.5.: Die Abbildung zeigt einen nordpolarstereografischen Temperaturplot auf dem zweiten Modellhöhenniveau als Beispielplot einer visualisierten Größe aus den LIMA-Daten.

wird angestrebt, die vorhandenen Temperaturplots für die Strömungsvisualisierung mit unterlegten Temperaturen zu nutzen, falls möglich. Die Strömungsvisualisierung müsste in diesem Fall genau so, wie die schon vorliegenden Plots als polarstereografische Projektion visualisiert werden.

Für die Darstellung des Windes werden die Parameter Windrichtung und Windgeschwindigkeit benötigt. Sie werden aus dem Horizontalwind, der in die meteorologischen Größen Zonalwind (Ost- Westkomponente u) und Meridionalwind (Nord- Südkomponente v) zerlegt wird, berechnet. Die Berechnung der Windrichtung in einem Punkt ist in Abbildung 4.6 veranschaulicht. Der Richtungsvektor in dem jeweiligen Quadranten ergibt sich aus den Gleichungen der dazugehörigen Winkel in Abbildung 4.6. Die Winkelgleichungen resultieren wiederum aus den Wertezustandsbedingungen von (u) und (v). Die Windgeschwindigkeit, in Meter je Sekunde, errechnet sich aus dem Betrag des horizontalen Windvektors mit der Formel $\sqrt{(u^2 + v^2)}$. Ob die zeitliche Auflösung der LIMA-Ergebnisdaten von sechs Stunden für einen flüssigen Strömungsfilm ausreichen, muss untersucht werden wenn eine entsprechende Lösung zur Realisierung des Strömungsfilmes vorliegt. Falls die zeitliche Auflösung nicht ausreichend sein sollte, muss zwischen den LIMA-Datensätzen entsprechend interpoliert

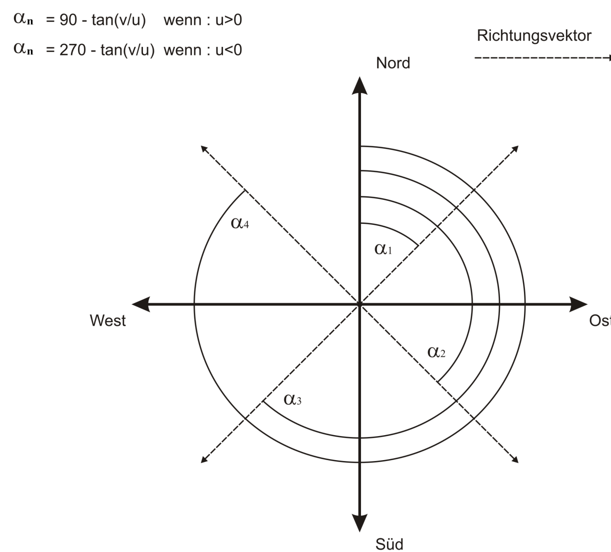

Abbildung 4.6.: Die Abbildung veranschaulicht das Prinzip der Berechnung der Windrichtung aus den Horizontalwindkomponenten u und v (aus [2]).

werden.

Wenn der Endpunkt einer Strömungslinie von ihrem Anfangspunkt aus für den ersten Zeitschritt berechnet wird, liegt dieser mit hoher Wahrscheinlichkeit nicht auf einem der Gitterpunkte des LIMA-Modells, die die benötigten Windgrößen punktuell enthalten. Für diesen Fall muss eine Wichtung, also wieder eine Interpolation, der nächsten umliegenden Gitterpunkte zum Endpunkt der Strömungslinie stattfinden. Das bewirkt, dass die Strömungslinien in den interpolierten Datensätzen ihre Richtung in Abhängigkeit der Wertigkeit und des Abstandes zu den sie umgebenden Gitterpunkten ändern. Bei fünf Zeitschritten sollte sich die Bewegung einer Strömungslinie in Abhängigkeit von vier Nachbargitterpunkten in etwa wie in Abbildung 4.7 (a) verhalten.

Für die Darstellung einer Strömungslinie muss die räumliche Struktur des Modells beachtet werden, die auf einer Kugel basiert. Das bedeutet, dass auch die Berechnungen der Strömungslinien für die Visualisierung, auf einer dreidimensionalen Kugel basieren müssen. Wenn z.B. an einem Punkt *A* die Windrichtung und die Windgeschwindigkeit bekannt sind, kann für einen bestimmten Zeitschritt, die darzustellende Strecke ermittelt werden. Diese Strecke darf nicht einfach als gerade Linie in das

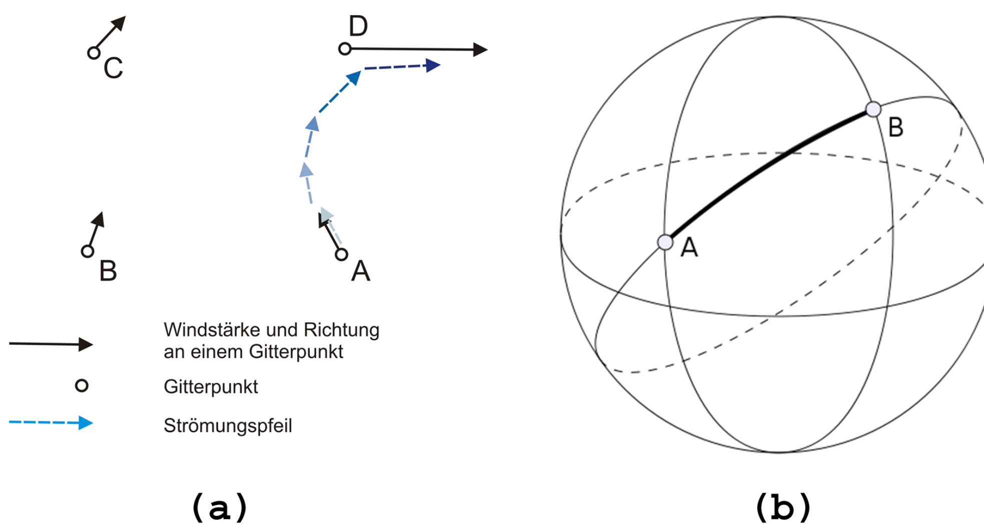

Abbildung 4.7.: Abbildung (a) veranschaulicht die Bewegung einer Strömungslinie in Abhängigkeit von vier Gitterpunkten (aus [2]) und (b) die kürzeste Verbindung zweier Punkte auf einer Kugel [23].

3D-Modell gezeichnet werden. Um die exakte Position des Endpunktes B der Linie zu bestimmen, muss sie als Orthodrome, also als kürzeste Verbindung zweier Punkte auf einer Kugeloberfläche berechnet werden. Anschließend kann für eine vereinfachende Darstellung die Verbindung der beiden Punkte als gerade Linie im Raum erfolgen, wenn der Abstand der Punkte nicht zu groß ist. Angenommen der Abstand der beiden Punkte ist etwa so groß wie in Abbildung 4.7 (b), dann sollte die Verbindung nicht anhand einer Sehne im Einheitskreis, der durch die Orthodrome aufgespannt wird, erfolgen. Diese Vorgehensweise würde den Weg der Strömungslinie in der Visualisierung massiv verfälschen. In so einem Fall sollten so viele Verbindungspunkte auf der Orthodrome berechnet werden, dass bei ihrer Verbindung mit geraden Linien optisch eine Kurve entsteht. Wird die Bestimmung des Endpunktes B nicht durch die Berechnung der Orthodrome ermittelt, sondern ausschließlich durch Zeichnen einer geraden Linie, die der Windstrecke des zu visualisierenden Zeitintervalls entspricht, würde die visualisierte Strömungslinie dem realen Windfeld nach wenigen Zeitintervallen davon laufen. Für die Realisierung der Strömungsvisualisierung müssen die Koordinaten der Strömungslinienabschnitte für jeden visualisierten Zeitschritt, unter Beachtung der oben genannten Methoden, berechnet werden. Die Werte, die die Grundlage für die Berechnung der Koordinaten bilden müssen ebenfalls für jeden visualisierten Zeitschritt, in Abhängigkeit von ihrer geografischen Umgebung neu berechnet werden.

Für die Umsetzung des Strömungsfilmes müssen die zu visualisierenden Daten aus den LIMA-Ergebnisdaten ausgelesen werden. Ihre Eigenschaften wurden bereits im Abschnitt 4.1.2 beschrieben. Sie stehen auf dem zentralen Fileserver des IAP, unter Beachtung der Bedienhinweise aus Abschnitt 2.2.3, zur Verfügung.

Für die Umsetzung der Aufgabenstellung steht als Programmiersprache IDL zur Verfügung. Da der Vorgabefilm der Universität Basel ebenfalls mit IDL realisiert wurde und es am IAP auf allen Rechnern und Großrechnern zur Verfügung steht, ist IDL das favorisierte Programmierwerkzeug mittels dem die Möglichkeiten zur Umsetzung der Aufgabenstellung untersucht werden sollen. Zudem sind mit IDL umfangreiche Erfahrungen bei der Realisierung verschiedenster Problemstellungen gesammelt worden, die bei der Erfüllung der Aufgabenstellung von Nutzen sein können.

4.4. Umsetzungsmöglichkeiten mit IDL

In den folgenden Abschnitten wird untersucht, welche unterschiedlichen Möglichkeiten IDL für Visualisierungen zur Verfügung stellt und wie sie für die Problemstellung genutzt werden können, die sich aus der Umsetzung der Aufgabenstellung ergeben.

IDL stellt in seiner Programmierumgebung sehr viele hilfreiche und effektiv arbeitende Funktionen und Prozeduren für die Verarbeitung unterschiedlichster Daten zur Verfügung. Zusätzlich wird eine Vielzahl von Funktionen und Prozeduren in den Umgebungen der Grafiksysteme bereitgestellt. Standardvisualisierungen können durch sie schnell und effizient erzeugt werden ohne dem Nutzer ein großes Hintergrundwissen abzuverlangen. Fortgeschrittenen Nutzern mit entsprechendem Hintergrundwissen bietet IDL individuelle Visualisierungsmöglichkeiten.

4.4.1. Grafiksysteme

IDL ist hauptsächlich auf die Visualisierung verschiedenster Problemstellungen in der Wissenschaft und Technik ausgerichtet. Es wird auf der offiziellen Firmenwebseite [24] damit geworben, dass mit einem geringen Programmieraufwand komplexe Daten verarbeitet, visualisiert und analysiert werden können. Das wird erreicht, indem IDL sehr viele unterschiedliche, vorgefertigte Visualisierungswerkzeuge besitzt, die auf die unterschiedlichsten Spezialgebiete zugeschnitten sind. Zum Teil können Visualisierungswerkzeuge mit grafischer Benutzeroberfläche auch komplett ohne Programmierung genutzt werden. Diese Visualisierungswerkzeuge ermöglichen es dem Anwender komplexe Visualisierungen zu erzeugen ohne tiefergehende Programmierkenntnisse zu besitzen oder sich intensiv mit Darstellungstechniken auseinandersetzen zu müssen.

IDL stellt dem Nutzer zwei grundlegende Grafiksysteme zur Verfügung, die er nur unabhängig voneinander, also nicht in Kombination miteinander, nutzen kann. Dass ist zum Einen, das *IDL Direct Graphics System*, das eine umfangreiche Programmbibliothek besitzt, zu denen auch die Visualisierungswerkzeuge gehören. Es ist für Nutzer entwickelt worden, die schnell und effektiv ihre Daten visualisieren möchten. Zum Anderen stellt IDL das *IDL Object Graphics System* zur Verfügung.

Die beiden Grafiksysteme können im Bezug auf die Visualisierung als unterschiedliche Programmiermethoden angesehen werden. Während die Programmierung im *IDL Direct Graphics System* einen sehr hohen Abstraktionsgrad besitzt, der es dem Nutzer sehr einfach machen soll zu seinem gewünschtem Ergebnis zu kommen, ist die Programmierung mit dem *IDL Object Graphics System* in Bezug auf die Höhe der Abstraktion, eine Stufe weiter unten angesiedelt. Beim Einsehen der Quellcodes der vorgefertigten Visualisierungswerkzeuge, wird klar, dass sie mittels des *IDL Object Graphics System* implementiert worden sind.

Bei der Programmierung mit dem *IDL Direct Graphics System* werden vom Nutzer die vorgefertigten Visualisierungswerkzeuge als Prozedur aufgerufen. Ihnen müssen die zu visualisierenden Daten in vordefinierter Weise übergeben werden. Zusätzlich können jedem Visualisierungswerkzeug eine Vielzahl von Parametern übergeben werden, die die Visualisierung individuell an die Problemstellung anpassen. Alle veränderbaren Parameter einer Prozedur werden ausführlich in der IDL-online-Hilfe beschrieben, was den Einstieg und das Arbeiten mit IDL sehr erleichtert. Die visualisierten Objekte werden hier automatisch positioniert, dem Ausgabemedium angepasst und gerendert. Die Dimensionen der Visualisierung und Transformierung der Daten auf die Darstellungsräume werden automatisch von den Prozeduren anhand von übergebenen Schlüsselworten durchgeführt. Das *IDL Direct Graphics System* ist eine leistungsstarke Programmierumgebung, die sehr mächtige Prozeduren zur Verfügung stellt, die wiederum den Programmieraufwand sehr gering halten und trotzdem ansprechende Standardvisualisierungen erzeugen. Die Ausgabe der Visualisierung kann auf eines der vordefinierten Ausgabegeräte wie z.B. Bildschirm, Drucker oder PostScript-Datei geleitet werden. Die Vielfalt der im *IDL Direct Graphics System* erzeugbaren Datenprojektionen und Visualisierungsarten hat natürlich seine Grenzen.

Um auch individuellen nicht standardisierten Visualisierungen gerecht zu werden, kann mit IDL auf das *IDL Object Graphics System* zurückgegriffen werden. Leider wird die Erstellung von Visualisierungen mit diesem Grafiksystem in wenig Literatur ausführlich behandelt. In den aktuell vorliegenden offiziellen IDL-Büchern ([25],[26],[27]) wird das Thema nicht behandelt oder darauf verwiesen, dass für dieses umfangreiche IDL Grafiksystem ein extra Buch geschrieben werden müsse. Vorliegende Quellen zum Thema sind in [28], einschlägigen Webseiten, wie z.B. auf der von David Fanning ([29]) und natürlich der IDL-online-Hilfe zu finden. Das *IDL*

Object Graphics System ist eine Sammlung von vordefinierten Objektklassen. Jede Objektklasse kapselt eine spezielle visuelle Repräsentation in sich. Tätigkeiten, wie das Modifizieren von Attributen, können mit Instanzen dieser Objektklassen durch das Aufrufen von vordefinierten Methoden durchgeführt werden. Diese Objekte sind für die Erstellung komplexer dreidimensionaler Datenvisualisierungen geschaffen worden. Im *IDL Object Graphics System* hat der Programmierer die volle Kontrolle über z.B. die Perspektive, Größe oder Position seiner Visualisierung. Er muss diese allerdings auch mit dem entsprechenden höheren Programmieraufwand komplett selbst entwerfen. Die verwendbaren Grafikobjekte unterliegen einer festgelegten Hierarchie (s. Abb. 4.8). Laut der IDL-online-Hilfe kann angeblich eine unbegrenzte Zahl an Grafikelementen in einem Modellobjekt zusammengefasst werden. Wobei spätestens der begrenzte Hauptspeicher des Rechners, auf dem die Visualisierung berechnet wird, die Anzahl der Grafikelemente begrenzt. In einem View(dt. Ansicht)-Objekt können wiederum mehrere Modellobjekte zusammengefasst werden und eine Szene kann mehrere Ansichten enthalten. Die Transformation der Grafikelemente wie z.B. Rotation, Skalierung oder Positionsveränderung werden über die Eigenschaften des *IDLgrModel object* kontrolliert, das einen dreidimensionalen Zeichenraum bereit stellt. Die Größe des Sichtfeldes auf das oder die Modelle, der verwendete Typ der Projektion, die Position des betrachtenden Auges und das dreidimensionale betrachtete Raumvolumen können über das *IDLgrView object*

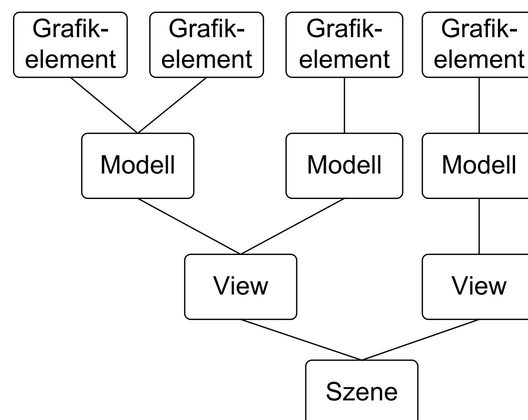

Abbildung 4.8.: Hierarchische Baumstruktur der Objekte im *IDL Object Graphics System*.

kontrolliert werden. Die Hierarchie legt fest, wenn an einem übergeordneten Objekt etwas verändert wird, alle untergeordneten Objekte im gleichen Maße mit verändert werden. Wird z.B. durch das Szeneobjekt in Abbildung 4.8 die Hintergrundfarbe geändert, gilt dies auch für alle Ansichtsobjekte, Modellobjekte und Grafikelemente. Ein Bild wird immer über ein *IDLgrView object* gerendert, also in der vorher definierten Ansicht sichtbar gemacht. Dieses Bild kann an ein Ausgabeobjekt wie z.B. Drucker, Fenster oder Buffer übergeben werden, über das es dann endgültig gespeichert oder ausgegeben wird.

4.4.2. Strömungslinien

Die Strömungslinie soll das grafische Element sein, das den Wind in der Visualisierung darstellt. Sie besteht aus mehreren Einzelteilen, die jeweils die zurückgelegte Strecke eines Partikels im Wind, für einen festen Zeitschritt markieren soll (s. Abb. 4.7). Diese Teilstrecken können mit Polygonen dargestellt werden, die in beiden Grafiksystemen zur Verfügung stehen. Polygone sind einfache geometrische Objekte, die aus den Grundbausteinen der Computergrafik, Punkte und Strecken, bestehen. Punkte werden durch die Angabe ihrer Koordinaten im zwei- oder dreidimensionalen Raum definiert. Strecken oder Linien werden wiederum durch die Koordinaten ihrer Endpunkte im Raum, z.B. P1(x_1,y_1,z_1) und P2(x_2,y_2,z_2), definiert. Ein Polygonzug besteht aus einer Aneinanderkettung von Strecken, er wird durch die Angabe seiner Eckpunkte, also der jeweiligen Endpunkte der einzelnen Strecken definiert. Der erste Punkt wird als Startpunkt und der letzte als Endpunkt bezeichnet. Um ein Polygon handelt es sich, wenn der Polygonzug geschlossen ist, also wenn der Startpunkt gleich der Endpunkt ist.

Im Quellcodebeispiel 4.1 wird gezeigt, wie im *IDL Direct Graphics System* Polygone mit dem Aufruf der Prozedur „`POLYFILL`“ erstellt werden können. Dabei müssen

Listing 4.1: Einfaches Polygon im *IDL Direct Graphics System* erzeugen.

```
x = [30, 100, 100, 30]
y = [30, 30, 100, 100]

POLYFILL,x,y, COLOR = 175,/DEVICE
```

ihr die Koordinaten x, y und eventuell z, der Eckpunkte als Vektoren übergeben werden. Mit dem Schlüsselwort „`COLOR`“ wird der Prozedur der Indexwert, der aktuellen Farbtabelle übergeben, mit dessen Farbe das Polygon gefüllt werden soll. Wenn mit diesem Schlüsselwort kein Farbwert übergeben wird, wird das Polygon mit einer vom System festgelegten Farbe gefüllt. Das Grafik-Schlüsselwort „`/DEVICE`“ stellt sicher, dass das Polygon in das Koordinatensystem des zuvor definierten Ausgabegeräts gezeichnet wird.

Ein Schlüsselwort, das die Transparenz des Polygons zu seinem Hintergrund beeinflussen kann, gibt es nicht. Im gesamten *IDL Direct Graphics System* wird Transparenz gar nicht unterstützt. Als einzige Alternative, wie sie im Quellcodebeispiel 4.2 gezeigt wird, gibt es für Polygone die Möglichkeit Transparenz zu imitieren. Im vorliegenden Beispiel wird eine Transparenz von 50 % imitiert. Dazu wird in Zeile

Listing 4.2: Möglichkeit der imitierten Transparenz im *IDL Direct Graphics System*.

```
1  ; die Pattern definieren
2  PAT1 = BYTARR(2,2)
3  ; Farbe der Patternmaske festlegen
4  PAT1[1,0] = 250
5  PAT1[0,1] = 250
6  PAT1[0,0] = 175
7  PAT1[1,1] = 175
8
9  ; Vektoren mit den X und Y-Werten festlegen
10
11 X1 = [20, 220, 220, 20]   & Y1 = [50, 50, 70, 70]
12 X2 = [30, 100, 100, 30]   & Y2 = [30, 30, 100, 100]
13 X3 = [110, 200, 200, 110] & Y2 = [30, 30, 100, 100]
14 X4 = [30, 100, 100, 30]   & Y3 = [50, 50, 70, 70]
15
16 ; Polygone zeichnen und mit Farbe füllen
17 POLYFILL, X1, Y1, COLOR = 250,/DEVICE
18 POLYFILL, X2, Y2, COLOR = 175,/DEVICE
19 POLYFILL, X3, Y3, COLOR = 175,/DEVICE
20 POLYFILL, X4, Y4, COLOR = 175,/DEVICE,pat=PAT1
```

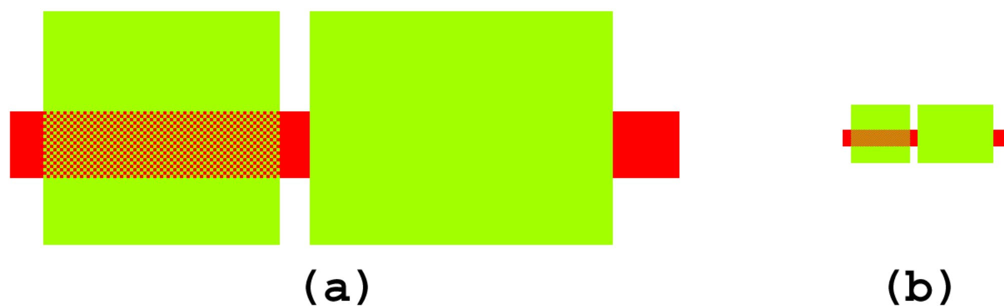

Abbildung 4.9.: Imitierte Transparenz im *IDL Direct Graphics System* bei Polygonen. Stark vergrößert in Bild (a), sodass die schachbrettartige Patternstruktur sichtbar wird. In gewünschter Größe im Bild (b), sodass das linke grüne Polygon, dem dahinter liegenden roten Polygon gegenüber anscheinend eine Transparenz von 50 % aufweißt.

zwei das Feld „`PAT1`“ der Dimension 2×2 generiert, das später als ein Pattern, also Muster, fungieren soll. In diesem Fall werden den Feldwerten in den Zeilen vier bis sieben abwechselnd die Farben rot und grün zugewiesen, um so ein Schachbrettmuster erzeugen zu können. In den Zeilen `11` bis `14` werden die Vektoren mit den Koordinaten der Eckpunkte der Polygone festgelegt. Darauf folgend werden in den Zeilen `17` bis `20` die Polygone der Reihe nach, wie sie in den Abbildungen 4.9 (a) und (b) zu sehen sind, gezeichnet. Bei der Zeichenreihenfolge der Polygone ist zu beachten, dass das zuletzt gezeichnete Polygon alle vorher gezeichneten an den Stellen verdeckt, an denen Überlappungen auftreten. Im vorliegenden Beispiel wird zuerst das lange rote Polygon, dann die beiden fast quadratischen grünen Polygone und als viertes das Polygon mit dem Muster gezeichnet. In Zeile `20` wird der „`POLYFILL`“ Prozedur das Feld mit dem Muster durch das Schlüsselwort „`pat`“ übergeben. Das letzte Polygon mit dem Muster wird dabei deckungsgleich der Überlappung des linken grünen Polygons mit dem roten positioniert, sodass in diesem Bereich eine halbdurchlässige Transparenz des grünen Polygons dem Roten gegenüber imitiert wird.

Diese Form der Transparenzimitation wird nur von der Prozedur „`POLYFILL`“ unterstützt. Der Effekt ist nur für die Überlappung von Flächen mit einheitlicher Farbe und zusätzlich einer Transparenzimitation von 50 % geeignet. Für die Realisierung der gewünschten Strömungslinie ist diese Methode demnach ungeeignet, da die Ergebnisse gegenüber einer wirklichen Transparenz eher unbefriedigend sind.

Im *IDL Object Graphics System* ist es möglich Polygone mittels von Polygon- oder Polyline-Objekten zu erstellen. Der Unterschied der beiden Polygonarten ist, dass bei der Generierung von Polygon-Objekten die Koordinaten der Eckpunkte übergeben werden müssen. Während Polyline-Objekten die Koordinaten der Verknüpfungspunkte einer zu zeichnenden Linie übergeben werden müssen. Mit den vielen zur Verfügung stehenden Schlüsselworten können die Eigenschaften der Objekte beeinflusst werden, wie z.B. die Breite einer Linie. Von besonderem Interesse ist, dass im *IDL Object Graphics System* für sehr viele Objekte ein beeinflussbarer Alphawert zur Verfügung steht, der für ein Maß der Transparenz des jeweiligen Objektes zu seinem Hintergrund steht.

Allgemein ist der Programmaufbau im *IDL Object Graphics System* etwas anders zu gestalten als im *IDL Direct Graphics System*. Hier muss z.B. die in Abschnitt 4.4.1 erwähnte Objekthierarchie beachtet werden, um zu einem Ergebnis zu gelangen. In IDL werden Objekte als Instanzen einer Klasse erschaffen, die in Form einer IDL-Struktur definiert ist. Der Name der Struktur ist also der Name der Klasse für die Instanz, das Objekt. Die Instanzdaten eines Objektes sind wiederum eine IDL-Struktur, die als Variable in einem *Objekt-heap* enthalten ist. Ihre Eigenschaften, *methods* genannt, können über spezielle Funktionen und Prozeduren beeinflusst werden. Um ein Objekt zu erstellen wird die Funktion „`obj_new`“ verwendet. In der Zeile

Listing 4.3: Objektumgebung für eine Visualisierung im *IDL Direct Graphics System*.

```
view = obj_new('IDLgrView')    ;View-Objekt erstellen
model = obj_new('IDLgrModel') ;Model-Objekt erstellen
view->add, model ;Modell-Objekt dem View-Objekt hinzufügen
    .
    .    ;Grafische Objekte dem Modell hinzufügen
    .

;Zeichenfenster erstellen
win = obj_new('IDLgrWindow', dimensions=[400, 200])
win->draw, view ;View-Objekt in Window-Objekt zeichnen
;Speicher für verwendete Objekte wieder freigeben
OBJ_DESTROY, model
OBJ_DESTROY, view
```

eins des Quellcodebeispiels 4.3 wird zu allererst ein „`View`“-Objekt erzeugt in dem die Visualisierung später gerendert werden soll. Das Objekt ist eine Instanz der Klasse „`IDLgrView`“. In der Zeile zwei wird das „`Model`“-Objekt erzeugt, in dem die gewünschten Grafikobjekte eingefügt und ausgerichtet werden sollen. Mit dem Ausdruck „`->add`“ wird dem „`View`“-Objekt das „`Model`“-Objekt, in der darauf folgenden Zeile, bekannt gemacht, es wird referenziert. In den Zeilen vier bis sieben folgt das eigentliche Einfügen und Ausrichten der grafischen Objekte wie z.B. Polygone, also das Visualisieren der Daten. Dies ist an dieser Stelle ausgeblendet, weil hier vorerst nur die benötigte Objektumgebung betrachtet wird, die für eine Visualisierung vorausgesetzt wird. In Zeile neun wird ein Ausgabeobjekt erstellt, in diesem Fall ein Bildschirmfenster, das 400 Pixel breit und 200 Pixel hoch ist. Mit „`->draw`“ wird die Ansicht des „`View`“-Objekts im Ausgabeobjekt gerendert.

Für alle im Programm erstellten Objekte werden Rechnerressourcen wie z.B. Hauptspeicher belegt, die auch nach Beendigung des Zeichnens, also dem Rendern, nicht automatisch wieder freigegeben werden. Selbst nach Beenden des Programms werden diese Ressourcen nicht wieder freigegeben, so lange die IDL Laufzeitumgebung noch im Hintergrund aktiv ist. Wenn z.B. ein Programm, eine Abfolge sich ändernder grafischer Objekte ausgeben soll, wird nicht die Position oder die Form der Objekte verändert, sondern es werden alle Objekte noch einmal neu erzeugt, ohne dass die vorherigen aus dem Speicher gelöscht werden. Dadurch kann es geschehen, dass die zugesicherten Hauptspeicherressourcen schnell ausgeschöpft sind und eine Auslagerungsdatei auf einer Festplatte zur Vergrößerung dieser Ressource angelegt wird. In diesem Fall wird das Programm extrem langsam. Um das zu verhindern, können bzw. sollten die Objekte zerstört und damit die belegten Ressourcen wieder freigegeben werden, bevor die Objekte erneut, z.B. in veränderter Position gezeichnet werden. Die Objekte können mit dem Aufruf „`OBJ_DESTROY`“ zerstört werden. Es kann jedes einzelne Grafikobjekt, wie einzelne Polygone, mittels der Prozedur „`OBJ_DESTROY`“ zerstört werden. Verfügt die Klasse eines Objektes über die Möglichkeit eine „`Cleanup`“ Prozedur aufzurufen wird dies automatisch bei Aufrufen der „`OBJ_DESTROY`“ Prozedur ausgeführt. „`OBJ_DESTROY`“ zerstört standardmäßig nur das Objekt an sich, nicht aber die enthaltenen Subobjekte. Die „`Cleanup`“ Prozedur bietet die Möglichkeit rekursiv alle Subobjekte, von denen das aufgerufene Objekt eine Referenz besitzt, zu löschen.

Diese sind z.B. für die Klassen „`IDLgrModel`“ und „`IDLgrView`“ vorhanden. Für

das Quellcodebeispiel 4.3 bedeutet das, dass im Prinziep nur Zeile `13` nötig wäre um alle Ressourcen wieder frei zu geben. Da ein „`View`“-Objekt mehrere „`Model`“-Objekte enthalten kann und es bei wiederholten Visualisierungen Sinn machen kann nach jeder Wiederholung die Elemente nur eines „`Model`“-Objekts wieder freizugeben, ist die Zeile `14` hier mit angegeben.

Das Quellcodebeispiel 4.4 zeigt wie ein Polygon-Linienobjekt erzeugt wird, das als sehr gute Möglichkeit für die Umsetzung der angestrebten Strömungslinie angesehen wird. In den Zeilen zwei bis fünf werden die Endpunktkoordinaten der Linie in den Vektoren „`x`“ und „`y`“ festgelegt. Die Zeilen sieben und acht gehören logisch zusammen, da sie zu lang sind, werden sie durch das „`$`“-Zeichen auf zwei Zeilen aufgeteilt. In diesen Zeilen wird das „`Polyline`“-Objekt aus der Klasse „`IDLgrPolyline`“ generiert. Dabei wird mit „`COLOR = 0`“ der Farbwert der Linie festgelegt. „`LINESTYLE = 0`“ legt fest, dass die Linie durchgezogen und „`Thick = 10`“ definiert wie breit sie dargestellt wird. Was die Darstellungsmöglichkeiten im *IDL Object Graphics System* so interessant für die Darstellung der Strömungslinie macht, ist die Möglichkeit die Transparenz verschiedenster Objekte beeinflussen zu können. Im Fall des „`Polyline`“-Objekts wird das in Zeile acht mit „`ALPHA_CHANNEL = 0.5`“ realisiert. Hier wird eine 50 % Transparenz festgelegt. Der Transparenzwert kann zwischen 0 und 1 variieren. Dabei gilt, je kleiner der Wert gewählt wird je transparenter ist das Objekt.

Um die Transparenz vor einem Hintergrund testen zu können, wird eine

Listing 4.4: Eine Polygon-Linie im *IDL Object Graphics System* erstellen.

```
1  ;Koordinaten der Endpunkte definieren
2  x[0] = -0.3
3  x[1] = -0.3
4  y[0] =  0.5
5  y[1] = -0.5
6  ;Polyline-Objekt erzeugen
7  mypolyline_1=OBJ_NEW('IDLgrPolyline',x,y, COLOR=0, $
8      LINESTYLE = 0,Thick=10,ALPHA_CHANNEL=0.5)
9  ;Polyline-Objekt in das Modell einfügen
10 model->add, mypolyline_1
```

Listing 4.5: Ein Hintergrundbild im *IDL Object Graphics System* laden und festlegen.

```
1 ;Hintergrundbild festlegen
2 background = read_image(file_which('meteor_crater.jpg'))
3 ;Hintergrundbild als Image-Objekt festlegen
4 image = obj_new('IDLgrImage', background);
5 ;Position und Skalierung des Image-Objektes festlegen
6 xc = [-1.0, 0.008]
7 image->setProperty, xcoord_conv=xc, ycoord_conv=xc
8 ;Image-Objekt dem Model-Objekt hinzufügen
9 model->add, image;Hintergrundbild
```

Pixelgrafik aus einer Datei geladen und der Visualisierung als Hintergrundbild hinzugefügt, wie es im Quellcodebeispiel 4.5 gezeigt wird. Die Pixelgrafik wird in der Zeile zwei ausgelesen und die Farbwerte der einzelnen Pixel im Feld „`background`“ festgehalten. Dabei erhält das Feld genau die Dimensionen, die das Bild besitzt. In Zeile vier wird das „`image`“, also das Bild-Objekt, von der Klasse „`IDLgrImage`“ erstellt. Der Vektor „`xc`“ enthält Skalierungsfaktoren, die die Koordinaten auf der jeweiligen Achse von Dateneinheiten in normalisierte Einheiten des „`image`“-Objekts konvertiert. Dabei verschiebt der erste Wert die Bildposition von der unteren linken Bildecke aus. Der zweite Wert steuert die Skalierung des Bildes. Das „`image`“-Objekt wird in Zeile acht dem Modell hinzugefügt. Das muss vor dem Hinzufügen der Linienobjekte zum Modell geschehen, da die Linien von der Reihenfolge her sonst hinter dem „`image`“-Objekt liegen würden und sie dann beim Rendern nicht sichtbar wären.

Die Abbildung 4.10 (a) zeigt die Visualisierung von drei „`Polyline`“-Objekten, mit je drei unterschiedlichen Transparenzwerten, vor einem Hintergrundbild. In dem Bild ist zu erkennen, dass der Hintergrund durch die horizontale und linke, vertikale Line durchscheint. Die Transparenzwerte der einzelnen Linien im Bild betragen „`0.9`“ für die rechte vertikale Linie, „`0.3`“ für die horizontale und „`0.5`“ für die linke vertikale Linie. In Bild (a) fällt auf, dass die Linien sich offensichtlich gegenseitig ignorieren. Das äußert sich darin, dass wenn eine Linie mit einem Transparenzwert eine andere überzeichnet, die untere keine sichtbaren Auswirkungen auf die obere Linie hat. Dieser Zusammenhang kann sehr gut in Abbildung 4.10 (a), an den Stellen, an denen sich die Linien kreuzen, nachvollzogen werden. Abbildung (b), in der die gleichen Linien, mit den jeweils selben Transparenzwerten vor einem weißen Hintergrund zeigt,

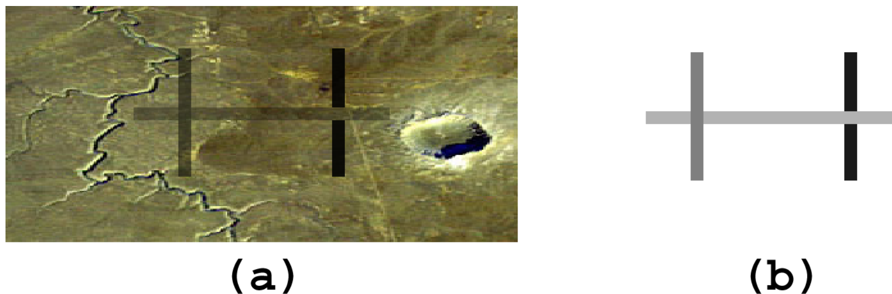

Abbildung 4.10.: Im Bild (a) wird Transparenz im *IDL Object Graphics System* bei Polygonen zu einem Hintergrundbild dargestellt. Im Bild (b) sind die Gleichen Polygon-Linien, mit denselben Transparenzwerten wie in (a) dargestellt. In (b) wird deutlich das der Transparenzeffekt der Polygone zueinander nicht funktioniert.

bestätigt diesen Effekt. Dabei wurde die unterste, also die rechte vertikale Linie dem Modell als letztes und die oberste, linke vertikale Linie dem Modell als erstes hinzugefügt. Die Polygonlinien werden also in der Reihenfolge, in der sie dem Modell hinzugefügt werden, von der Betrachtungssicht aus, von vorne nach hinten angeordnet.

Für die Umsetzung der gewünschten Strömungslinie scheint das *IDL Object Graphics System* die geeigneten Mittel zur Verfügung zu stellen. Sie kann z.B. durch eine Aneinanderkettung von mehreren Polygonlinien, die eine zunehmende Transparenz zum Ende hin besitzen, dargestellt werden, wie es in der Abb. 4.11 zu sehen ist. Durch die zur Verfügung gestellten Polygonlinien kann z.B. die einheitliche Breite der Linien sichergestellt werden. In wie fern sich der Effekt, dass sich die Polygonlinien gegenseitig verdecken, als negativ herausstellen könnte, muss untersucht werden wenn eine Umsetzung des Strömungsfeldes realisiert ist.

Für die Realisierung der Strömungslinie wurden auch Ansätze untersucht, sie ohne direkten Transparenzeffekt umzusetzen. Sie werden hier aber nicht weiter erwähnt, weil sie ein unbefriedigenderes Resultat liefern und es den gesetzten Rahmen dieser Arbeit übersteigen würde.

Abbildung 4.11.: Das Bild zeigt eine Möglichkeit die Strömungslinien (weiss) mit der Aneinanderkettung von Polygonlinien und ihrer zunehmenden Transparenz zum Ende hin zu realisieren.

4.4.3. Temperaturplot

Bei Internetrecherchen zur Untersuchung von möglichen Transparenzeffekten in Darstellungen, die mit IDL erzeugt werden, ist auf der Webseite von D. Fanning [29] eine Möglichkeit gefunden worden, Pixelgrafiken so ineinander zu rechnen, dass ein Transparenzeffekt entsteht. Diese Methode findet auf der Ebene von Feldberechnungen statt und ist deshalb von den beiden IDL Grafiksystemen unabhängig. In Quellcodebeispiel 4.6 wird das Prinzip dieser Methode dargestellt. In den Zeilen zwei und vier werden die beiden Pixelgrafiken zunächst in die Felder „`cloudp`“ und „`earthp`“ geschrieben. Die Bilder besitzen beide die gleichen Dimensionen in Höhe und Breite. Diese Dimensionen werden für die beiden Felder vom Typ BYTE automatisch beim Auslesevorgang übernommen. Jedes Feldelement der Felder „`cloudp`“ und „`earthp`“ besitzt nun den Farbwert eines Pixels der beiden Bilder. In Zeile sechs wird der Transparenzfaktor des oben liegenden Bildes festgelegt. Er darf Werte zwischen „0“ und „1“ annehmen. Das oben liegende Bild ist im Beispiel im Feld „`cloudp`“

Listing 4.6: Grafiksystem unabhängige Transparenz erzeugen.

```
1 ;erstes Bild auslesen
2 cloudp = read_image(file_which('cloud.jpg'))
3 ;zweites Bild auslesen
4 earthp = read_image(file_which('earth.jpg'))
5 ;Transparenzfaktor
6 alpha = 0.5
7 ;Bilder ineinander rechnen
8 rgb = BYTE(alpha*FLOAT(cloudp)+(1.0-alpha)*FLOAT(earthp))
```

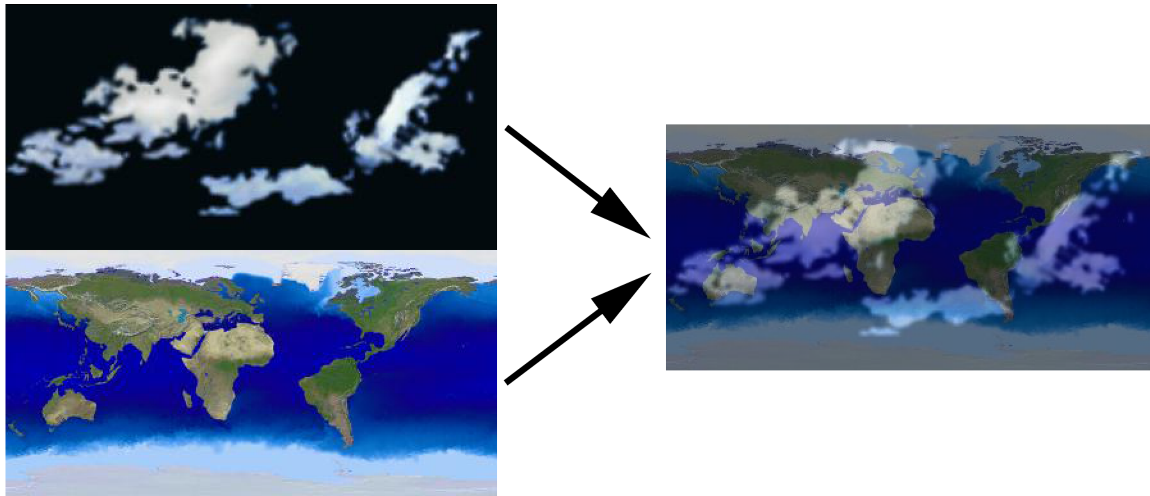

Abbildung 4.12.: Das Bild zeigt das Ergebnis des Quellcodebeispiels 4.6. Links sind die beiden Originalbilder übereinander abgebildet, rechts ist das Resultat zu sehen, in dem die beiden Bilder verrechnet sind.

enthalten. Die Verrechnung der beiden Bilder findet in Zeile acht statt. Hierbei handelt es sich um eine lineare Interpolation zwischen den Feldwerten der beiden Felder mit denselben Feldindizes. Es werden also, wenn die Bilder exakt übereinander liegen, die Farbwerte der sich gegenüberliegenden Pixel linear interpoliert (s. Abb. 4.12).

Diese Methode würde den Einsatz der schon vorhandenen Temperaturplots des LIMA-Modells ermöglichen, wenn sichergestellt wird, dass bei der Bildinterpolation die dargestellten Daten nicht oder vertretbar gering verfälscht werden. Dazu wurden die Einzelbilder einer Visualisierung von Eisdaten herangezogen, deren originale Ergebnisdaten vom Modell in einer zeitlichen Auflösung von einer Stunde gespeichert wurden. Für die Visualisierung in einem Film wurden die Originaldaten in einer zeitlichen Auflösung von 12 Minuten, also vier Zwischenbilder je voller Stunde, mit einem Spline-Algorithmus interpoliert und anschließend in Einzelbildern visualisiert. Zum Vergleich wurden die fertiggestellten Visualisierungen der Eisdaten zu jeder vollen Stunde, die den Originaldaten entsprechen, ausgewählt und mit der zuvor beschriebenen Bildinterpolationsmethode vier Zwischenbilder generiert. Anschließend sind die beiden, jeweils der gleichen Uhrzeit entsprechenden Visualisierungen nebeneinander zu einem Bild zusammengefügt worden, um einen direkten Vergleich ziehen zu können. Das Ergebnis fällt überraschend besser aus als erwartet. Der Vergleich der Einzelbilder, wie in Abbildung 4.13 zu sehen, zeigt, dass zwischen den beiden

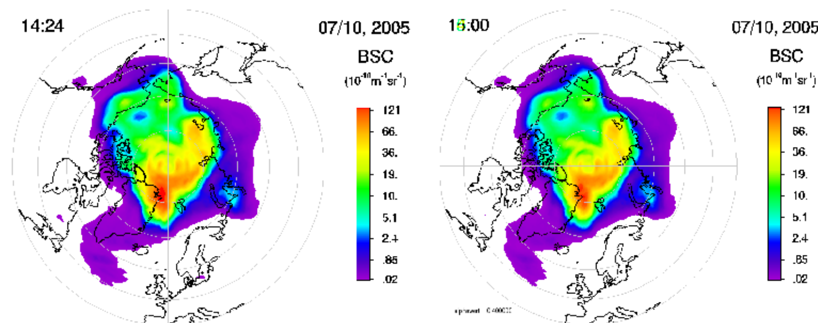

Abbildung 4.13.: Das Bild zeigt links eine Visualisierung von Eisdaten die viermal innerhalb einer Stunde, mit einem Spline-Algorithmus interpoliert wurden. Rechts ist die gleiche Visualisierung zu sehen, die anhand der Bildinterpolation aus zwei Bildern zur vollen Stunde erstellt wurde.

Bildern in den visualisierten Daten nur sehr geringe Unterschiede auftreten. Im Bild ist links das interpolierte und anschließend visualisierte Datenfeld zu sehen. Rechts ist die Version zu sehen, die aus der Bildinterpolation hervorgeht, erkennbar an der falschen Zeitangabe oben links in der Visualisierung. Die Bilder zeigen das Ergebnis der zweiten von vier Zwischeninterpolationen der vollen Stunde. Durch die sequenzielle Abfolge der Einzelbilder im Film ist im direkten Vergleich in den visualisierten Daten kein Unterschied mehr erkennbar.

Das Problem der sich ständig verändernden Uhrzeit oben links in der Visualisierung ist auch im Temperaturplot der LIMA-Daten vorhanden. Da das Problem nur Bilddaten betrifft, die einen schwarzen oder weißen Farbwert besitzen, ist eine Möglichkeit das Problem zu lösen, diese Daten, wenn sie in den Feldern vorliegen, zu isolieren und von der Interpolation auszuschließen. So wird die Uhrzeit der vollen Stunde in allen nachfolgend interpolierten Bildern angezeigt, bis eine neue anbricht. Diese Art der Lösung des Problems ist nur möglich, wenn die Farbwerte, mit denen die Schriftzeichen in der Visualisierung dargestellt sind, nicht in den visualisierten Daten vorkommen.

Die Untersuchung der Bildinterpolation hat ergeben, dass die sichtbare Verfälschung der visualisierten Daten vertretbar gering ist und sie für die Erstellung von Zwischenbildern im Strömungsfilm geeignet ist. Durch die Möglichkeit der Nutzung, der vorhandenen Temperaturvisualisierung wird ein hoher Programmieraufwand zur

erneuten Visualisierung und damit auch ein hoher zeitlicher Aufwand verhindert.

Es sollte in diesem Fall aber beachtet werden, dass die Strömungsvisualisierung in derselben polarstereografischen Projektion erfolgen muss. Die Temperaturdaten sind bei ihrer ursprünglichen Visualisierung auf eine dreidimensionale Kugel projiziert und die Ansicht auf den jeweiligen Pol zentriert worden. Dadurch entsteht in der Visualisierung eine Verdichtung der Daten im zunehmenden Maße zum Äquator hin. Bei der Überlagerung der Strömungsdaten über die Temperaturplots, müssen diese genau so, wie ursprünglich die Temperaturdaten auf eine dreidimensionale Kugeloberfläche projiziert werden. Damit wird sichergestellt, dass bei einer Kombination von Wind und Temperaturdaten, diese auch an den geografisch korrekten Positionen dargestellt werden.

4.5. Praktische Umsetzung des Strömungsfilms

In den folgenden Abschnitten wird die praktische Realisierung des Strömungsfilms beschrieben. Die programmiertechnische Umsetzung wird teilweise anhand von exemplarisch gewähltem Quellcode erfolgen.

4.5.1. Schematische Umsetzung, funktioneller Ansatz

Die Strömungsvisualisierung, in der Winddaten und Temperaturdaten kombiniert werden, ist in einem IDL-Programm umgesetzt worden. Das Programm wurde auf einem PC, der im IAP-Netzwerk integriert ist und Zugriff auf die IAP-Rechnerressourcen besitzt, entwickelt. Bei der Entwicklung ist von Anfang an, z.B. bei der dynamischen Lade- bzw. Speicherpfadgenerierung, darauf geachtet worden, dass das Programm mit verhältnismäßig wenig Aufwand auf einem Abteilungsserver eingepflegt werden kann. Es beschränkt sich auf die Erstellung von Einzelbildern, da Erfahrungen gezeigt haben, dass eine Filmerstellung mit IDL nur in sehr eingeschränktem Maße möglich ist. Für die Filmerstellung aus Einzelbildern sind verschiedene kommerzielle sowie freie Programme verfügbar, die es möglich machen mit geringem Aufwand zu einem gewünschten Ergebnis zu gelangen. Die am IAP genutzte Software für diese Aufgabe

ist Convert und MJPEGTOOLS für die Linux Betriebssysteme auf den Abteilungsrechnern und VideoMach für die Arbeitsplatzrechner mit Windowsbetriebssystemen.

Funktionell ist das Programm wie folgt aufgebaut. Zuerst werden die Koordinaten aller Gitterpunkte des LIMA-Modells aus einer Datei gelesen. Mit diesen Daten und der Angabe, wie viele Strömungslinien in der Visualisierung auftreten sollen, wird dann ein Feld generiert, in dem die Initialisierungspunkte aller Strömungslinien festgehalten sind. Daraufhin beginnt eine äußere Schleife, in der die LIMA-Ergebnisdateien und die daraus generierten polarstereografischen Temperaturplots, für die zeitliche Modellauflösung von sechs Stunden ausgelesen werden. Dabei wird auf die verschiedenen Server zugegriffen, auf denen die entsprechenden Daten vorliegen, wie es im Datenflussschema (Abb. 4.14) veranschaulicht ist. In dieser Schleife folgt dann eine innere Schleife, in der die gewünschte Anzahl von

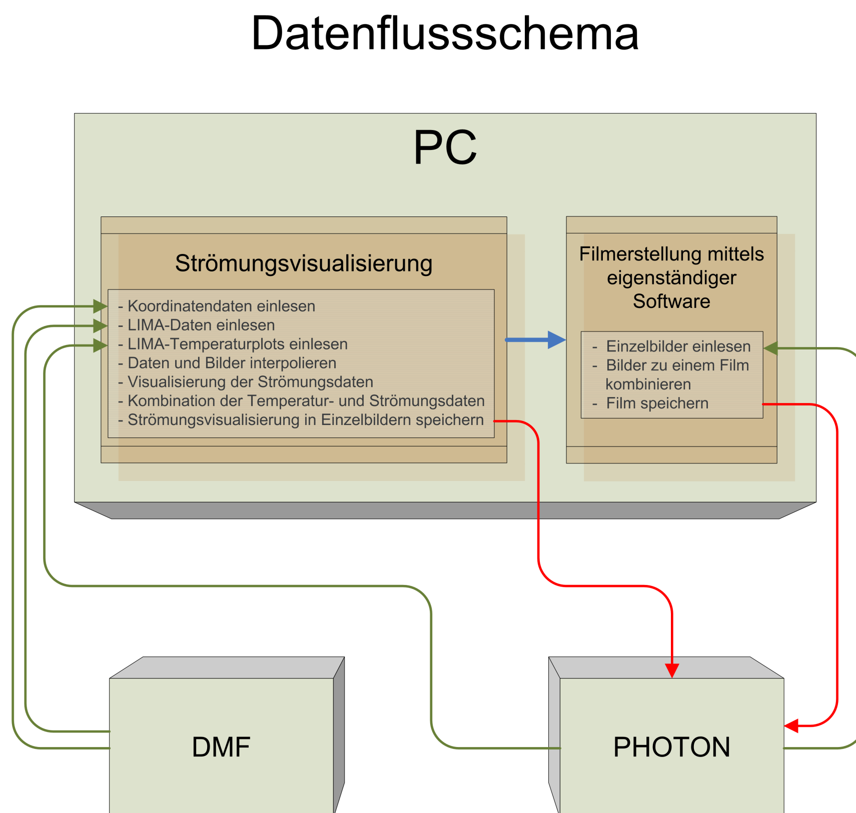

Abbildung 4.14.: Das Bild zeigt das Datenflussschema der aktuellen Strömungsvisualisierung.

Interpolationsschritten in der sechs Stunden-Auflösung ausgeführt werden. Interpoliert wird hier zwischen den Bitmapgrafiken der LIMA-Temperaturplots in der Weise, wie sie in Abschnitt 4.4.3 beschrieben ist und den auf die benötigten Winddaten reduzierten LIMA-Ergebnisdaten. Die für den aktuellen Zeitschritt interpolierten Daten werden in einer Übergabestruktur festgehalten und anschließend zwei aufgerufenen Unterprogrammen übergeben. Im ersten Unterprogramm wird nach den nächsten Gitterpunktnachbarn des aktuell letzten Wegpunktes jeder Strömungslinie gesucht. Aus den Windwerten dieser Gitterpunkte werden die Windwerte des zuvor beschriebenen Wegpunktes errechnet und in der Übergabestruktur gespeichert. Nach der Rückkehr in das Hauptprogramm wird ein zweites Unterprogramm aufgerufen, in dem anhand der neuen Winddaten des letzten Wegpunktes der Strömungslinien, die Koordinaten des nächsten Wegpunktes errechnet werden. Diese Werte werden daraufhin in der Übergabestruktur gespeichert, die Strömungslinien gezeichnet und wieder ins Hauptprogramm zurückgekehrt. Anschließend werden die Strömungslinien mit dem interpolierten Temperaturplot in einer Grafik kombiniert und als Einzelbild gespeichert.

In den folgenden Abschnitten wird auf die programmiertechnische Umsetzung der Strömungsvisualisierung genauer eingegangen.

4.5.2. Hauptprogramm

Der Name des IDL-Programms, das die Strömungsvisualisierung realisiert lautet „`windfeldhemisphaere_7.pro`“. Es liegt aktuell in der Entwicklungsversion sieben vor. Das Hauptprogramm beinhaltet, wie das Funktionsschema in der Abbildung 4.15 zeigt, die festzulegenden Steuerparameter der Visualisierung, die Definition der davon abhängigen Felder und Variablen, das Auslesen der benötigten LIMA-Daten und der Temperaturplots in einer äußeren Datenleseschleife und einer inneren Schleife, in der, die Daten und Bilder für die gewünschten Zeitschritte interpoliert werden. Die Interpolationsschleife beinhaltet außerdem die Aufrufe zweier Unterprogramme, die die Strömungsvisualisierung realisieren, grafische Operationen zur Kombination der visualisierten Daten, sowie das Speichern der endgültigen Visualisierung in Einzelbildern.

Die Anzahl der zu erstellenden Einzelbilder wird durch die

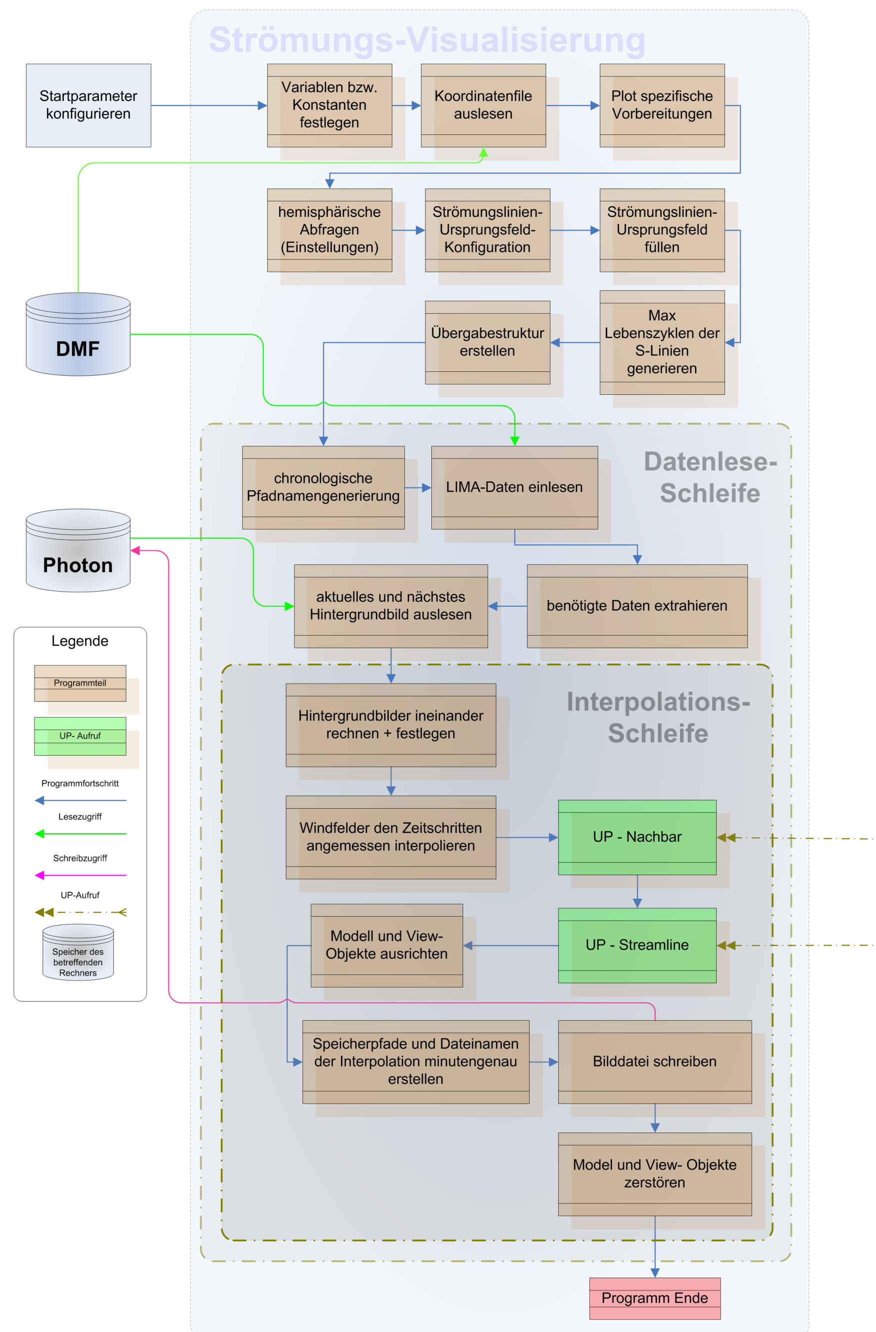

Abbildung 4.15.: Das Bild zeigt das Funktionsschema des Hauptprogramms der Strömungsvisualisierung.

Initialisierungseingabedaten Zeitraum und Zeitschritt, in denen die LIMA-Daten visualisiert werden sollen, festgelegt. Bei einem Zeitraum von zwei Tagen und einem Visualisierungszeitschritt von 15 Minuten, wie es im Quellcodebeispiel 4.7 gezeigt wird, ergibt sich daraus eine Einzelbildanzahl von 192. Für das Definieren des Zeitraums werden in dem Feld „`zeitgrenzen`“ vor dem Start des Programms, das Anfangs- und Enddatum, jeweils in Tages, Monats und Jahresangaben, festgelegt. Der Visualisierungszeitschritt wird in Sekunden, in der Variablen „`zeitschritt_s`“ festgehalten. Die LIMA-Ergebnisdaten und die Temperaturplots liegen, wie schon erwähnt, in einer zeitlichen Auflösung von sechs Stunden vor und ist in der Variablen „`timeresol`“ festgehalten. Weiterhin muss vor dem Programmstart die Hemisphärenansicht, wie in Zeile 14, gewählt werden. Da die anderen LIMA-Größen in derselben Art und Weise visualisiert worden sind wie die Temperaturen, kann das zu visualisierende Strömungsfeld bei Bedarf auch mit anderen Daten kombiniert werden. Dazu kann mit der Variablen „`val`“ die gewünschte Größe gewählt werden.

Im laufenden Programm wird darauf folgend die Anzahl der Zeitschritte in der LIMA-Auflösung von sechs Stunden errechnet, in diesem Fall 24 und in der

Listing 4.7: Primäre Steuerparameter des Programms.

```
1  zeitgrenzen = INTARR(6)
2
3  zeitgrenzen[0]= 2008     ;Anfangsdatum - jahr
4  zeitgrenzen[1]= 6        ;monat
5  zeitgrenzen[2]= 1        ;tag
6  zeitgrenzen[3]= 2008     ;Enddatum - jahr
7  zeitgrenzen[4]= 6        ;monat
8  zeitgrenzen[5]= 2        ;tag
9
10 timeresol = 6.           ;zeitliche Auflösung der Datenfiles
11 ;zeitschritt in s
12 zeitschritt_s = 900.     ;15 min
13 ;Auswahl fuer nord- und suedhemisphaere (0;1) 0=Sued/1=Nord
14 mhemisphere = 1
15 kakt = 2                 ;Höhe
16 val = 0                  ;Groesse (t=0,u=1,v=2,w=3)
```

Variablen „zeitschritt_i“ festgehalten, die für spätere Schleifensteuerungen benötigt wird. Anschließend werden Parameter deklariert, die wie z.B. der Erdradius und Erdumfang in Metern für die Berechnungen der Strömungslinien benötigt werden. Für das Auslesen der verschiedenen LIMA-Daten werden Variablen definiert, die die Dimensionen der Datenfelder besitzen, wie z.B. die Gesamtgitterpunktanzahl des 110er LIMA-Modellgitters (41804) oder die Anzahl der einzelnen atmosphärischen Höhenniveaus (118) sowie die Anzahl der Breitenringe auf dem Globus in Eingradabschnitten. Weiterhin werden hier für die Generierung der Dateinamen die Anzahlen der Tage in den Monaten eines normalen Jahrs und eines Schaltjahrs in einem Feld festgelegt.

Als erstes wird im Programm die Datei „anfcommon110.dat“ ausgelesen, in der die Daten gespeichert sind, anhand derer die Koordinaten der einzelnen Gitterpunkte errechnet werden können.

Im Anschluss werden, wie im Quellcodeauszug 4.8 gezeigt wird, die benötigten

Listing 4.8: Grafische Initialisierungen und Parameter.

```
;View-Objekt erstellen
view = obj_new('IDLgrView')
;Model-Objekt erstellen
model_i1 = obj_new('IDLgrModel')
;Modell-Objekt dem View-Objekt hinzufuegen
view->add, model_i1
;Model-Objekt erstellen
model_i2 = obj_new('IDLgrModel')
;Modell-Objekt dem View-Objekt hinzufuegen
view->add, model_i2
;Zeichenfenster erstellen
buff = obj_new('IDLgrBuffer', dimensions=[1024,768])
;X-Y-Achsenkorrektur der Daten gegenüber dem Hintergrundbild
x_achsenkorrektur = -0.129
y_achsenkorrektur = 0.042
;Position und Skalierung des Image-Objektes festlegen
xc = [-1.0, 0.002]
yc = [-1.0, 0.0026]
```

grafischen Objekte initialisiert und Parameter für die Darstellung in den Objekten festgelegt. In der Zeile zwei wird das „`view`"-Objekt initialisiert, indem die gesamte Visualisierung gerendert werden soll. In den Zeilen vier und acht werden die „`model`"-Objekte initialisiert, je eins für die Temperaturplots und eins für die Strömungsvisualisierung. Jeweils darauf folgend in den Zeilen sechs und zehn werden die „`model`"-Objekte dem „`view`"-Objekt hinzugefügt. Das gerenderte Bild soll einem Ausgabe-Objekt der Klasse „`IDLgrBuffer`" (s. Zeile `12`) übergeben und anschließend den Bilddimensionen der Temperaturplots von 1024 × 768 Bildpunkten angepasst, in einer Pixelgrafikdatei gespeichert werden. In den Zeilen `14` und `15` sind die Werte festgehalten, die benötigt werden um die Strömungsvisualisierung über den Temperaturdaten deckungsgleich zu positionieren. Die Vektoren in den Zeilen `17` und `18` beinhalten Parameter, die den Temperaturplot im „`image`"-Objekt in Abhängigkeit des „`model`"-Objekts und damit auch des „`view`"-Objekts so skalieren und positionieren, dass er die Sichtfläche unverzerrt ausfüllt.

Die definierten Eigenschaften einer Strömungslinie sind ihre Ursprungskoordinaten, ihr Lebenszyklus und ihre maximale sichtbare Länge in Zeitschritten. Die Ursprungskoordinaten definieren die Ausgangspunkte der einzelnen Strömungslinien. Von diesem Punkt aus beginnt die Strömungslinie in das Windfeld zu strömen, bzw. wird auf diese Koordinaten zurückgesetzt wenn sie ihren maximalen Lebenszyklus erreicht hat oder den Äquator überschreitet. Der Lebenszyklus beschreibt die maximale Anzahl an visualisierten Zeitschritten, die sich die Strömungslinie durch das Windfeld bewegen darf. Die maximale sichtbare Länge der Strömungslinie legt fest, nach dem wievielten Zeitschritt hinter der aktuellen Position die Strömungslinie nicht mehr gezeichnet wird. Die eigentliche sichtbare Länge im Sinne von Entfernung, hängt dadurch wiederum von der Windgeschwindigkeit ab, die durch die Strömungslinie angezeigt wird.

Im nächsten Schritt werden die Ursprungskoordinaten der einzelnen Strömungslinien definiert. Dazu werden zunächst die hemisphärenabhängigen Parameter ermittelt, bzw. definiert. Das sind z.B. die Indexwerte der ersten und letzten genutzten Breitenkreise, die Koordinaten des ersten Gitterpunktes des verwendeten Modellgitterabschnitts, sowie die entsprechenden Teil-Strings zur Dateinamengenerierung. Der maximale Lebenszyklus der Strömungslinie wird in Zeile eins des Quellcodeauszugs 4.9 festgelegt. Die maximale sichtbare Länge in Zeile zwei. Die Variable „`feldteiler`" erhält den Wert, der beschreibt, jeder wievielte Gitterpunkt, der Reihe nach einen

Listing 4.9: Definition des Strömungslinien-Ursprungsfeldes.

```
1 maxlive = 60  ;maximale Lebensdauer der Stroemungslinie
2 maxline = 40  ;maximale Laenge der Stroemungslinie
3 feldteiler = 17
4 urp_anzahl = FIX((igitnb_v/feldteiler)/2)
5 ;Ursprungsfeld der Strömungslinien für phi und lamda in RAD
6 ursprungskoordfeld = FLTARR(urp_anzahl,2)
```

Ursprungspunkt für eine Strömungslinie darstellt. In der darauf folgenden Zeile vier, wird die Anzahl der definierten Ursprungspunkte errechnet. Diese ergibt sich aus der Gesamtgitterpunktanzahl geteilt durch den „`feldteiler`“. Das Ergebnis wird wiederum durch zwei geteilt, da nur eine Hemisphäre gezeichnet wird. In der Zeile sechs wird anhand der errechneten Ursprungspunktanzahl dann das entsprechende Feld deklariert, in das für jeden Punkt die beiden Winkelkoordinaten Lambda und Phi auf der Einheitskugel, in der folgenden Schleife geschrieben werden.

Im darauf folgenden Programmabschnitt (s. Quellcodeauszug 4.10) werden Variablen und Felder deklariert und mit Werten gefüllt, die für die Strömungsvisualisierung benötigt werden. Das Feld „`streamline_field_s`“ stellt das Koordinatengedächtnis aller Polygonlinien dar, die die Strömungslinien darstellen sollen. Die erste Dimension steht für die Anzahl der Strömungslinien, die zweite für die jeweilige Koordinate in Lambda und Phi, der Wegpunkte und die dritte für die Anzahl der sichtbaren letzten Streckenabschnitte der Strömungslinie. Im Feld „`streamline_index_s`“, in Zeile zwei, soll im ersten Index die maximale Lebensdauer und im zweiten, wie viele Zeitschritte die jeweilige Strömungslinie schon lebt, festgehalten werden. Das Feld „`streamline_livecycle_s`“ speichert wieviele Zeitschritte die

Listing 4.10: Deklaration der Strömungslinienfelder.

```
1 streamline_field_s =      FLTARR(urp_anzahl,2,maxline)
2 streamline_index_s =      INTARR(urp_anzahl,2)
3 streamline_livecycle_s = INTARR(urp_anzahl)
4 streamline_ursprung_s =  FLTARR(urp_anzahl,2)
5 streamline_u_v_s =        FLTARR(urp_anzahl,2)
```

Strömungslinie insgesamt schon existiert. „`streamline_ursprung_s`“ beinhaltet das oben schon beschriebene Ursprungsfeld und „`streamline_u_v_s`“ soll die Werte der U- und V-Windkomponenten des zuletzt ereichten Wegpunktes der Strömungslinien enthalten. Im anschließenden Programmteil werden die unterschiedlich langen Lebenszyklen der Strömungslinien realisiert.

Die Definition von unterschiedlich langen Lebensdauern der einzelnen Strömungslinien ist bei dieser Visualisierung von wesentlicher Bedeutung, um ein möglichst gleichmäßiges, mit Strömungslinien gefülltes, Windfeld zu erzeugen. Wenn bspw. alle Strömungslinien die gleiche Lebenszyklusdauer besäßen, würden alle ständig zu einem gemeinsamen Zeitpunkt an ihren Ursprungspunkten entstehen und alle wieder gemeinsam irgendwo im Windfeld verschwinden. Dieser Effekt führt dazu, dass die Aussagekraft der Strömungsvisualisierung sehr eingeschränkt wird. Der Effekt tritt auch dann auf, wenn die Strömungslinien zwar unterschiedlich lange Lebenszyklusdauern besitzen, diese aber die gemeinsamen Teiler einer größeren Zahl darstellen. Wenn die unterschiedlichen Lebenszyklen z.B. 3, 10 und 15 Zeitschritte sind, werden alle Strömungslinien beim dreißigsten Zeitschritt gemeinsam verschwinden. Wird versucht dieses Problem zu umgehen, indem die Lebensdauer der Strömungslinien über die gesamte Zeit der Strömungsvisualisierung andauert, werden sich die Strömungslinien in den Hauptströmungen oder windstillen Zonen des im Fall der Erde unendlichen, bzw. randlosen Windfeldes, nach kurzer Zeit konzentrieren (s. Abb. 4.16).

Quellcodeausschnitt 4.11 zeigt die Realisierung der unterschiedlichen Lebenszykluslängen der einzelnen Strömungslinien. Dazu wird ein Feld mit ganzen Zufallswerten im Bereich von `0` bis `maxlive-20` gefüllt. Anschließend werden zu

Listing 4.11: Generierung der unterschiedlichen Lebenszyklen.

```
;Maximale Lebenszyklen der einzelnen linien generieren
rarr = intarr(urp_anzahl)            ;random array
;Array mit zufallszahlen zw. 0 und maxlive -20 fuellen
rarr[*]=RANDOMU(seed,urp_anzahl,/Uniform,/Double)*(maxlive-20)
;alle werte +20 (keine Null mehr!)
rarr[*]=(rarr[*]+20)
streamline_index_s[*,0] = rarr[*]
streamline_index_s[*,1] = 0
```

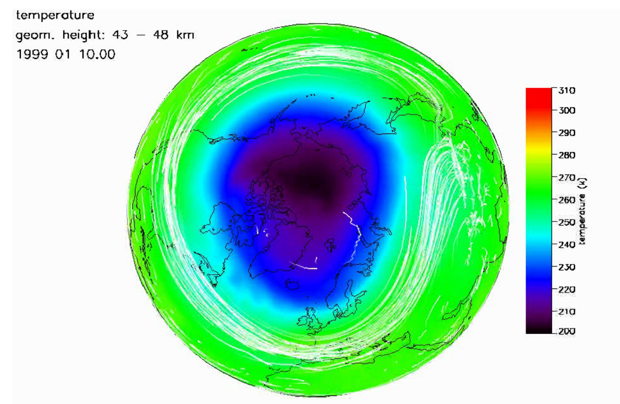

Abbildung 4.16.: Das Bild zeigt den Fall einer Strömungsvisualisierung, in dem die Lebensdauer der Strömungslinien eine längere Zeit anhält, sodass sie sich in Hauptströmungen und windstillen Zonen des Windfeldes konzentrieren.

allen Werten des Feldes `20` addiert, sodass der Wertebereich der Zufallszahlen dann zwischen `20` und `maxlive` liegt. Die verwendete Weise der unterschiedlichen Lebenszykluslängengenerierung hat bei der Testung verschiedener Ansätze die besten Resultate geliefert. Diese Werte werden dann in das Feld „`streamline_index_s`" geschrieben, das im folgenden Programmabschnitt einer Übergabestruktur zusammen mit allen anderen, für die Strömungsvisualisierung benötigten Variablen, Feldern und Objekten bekannt gemacht werden.

Im anschließenden Programmabschnitt beginnt die äußere Schleife, in der die LIMA-Daten und die Temperaturplots ausgelesen werden. Sie ist als „`while`"-Schleife implementiert und wird so lange wiederholt, bis der letzte Zeitschritt der sechs Stunden-Auflösung, der LIMA-Daten, im Enddatum erreicht ist. Als erster Schritt in der äußeren Schleife werden die Ladepfade und Dateinamen der LIMA-Ergebnisdateien und Temperaturplots erstellt. Dabei wird darauf geachtet, dass die Ladepfade exakt der Ordnerstruktur der Speicherorte auf dem DMF bzw. dem Photon entsprechen. Anschließend werden die LIMA-Ergebnisdaten, wie im Quellcodeauszug 4.12 gezeigt, ausgelesen. Die Daten sind im unformatierten Format angelegt. Für das fehlerfreie Auslesen der Daten ist es wichtig die Dimensionen und Datentypen aller Werte und Datenfelder zu kennen. Bei der Initiierung des Leseprozesses in Zeile eins, ist der gesamte Ladepfad mit Dateinamen in der Variablen „`datafile`" als String

Listing 4.12: Auslesen der LIMA-Daten.

```
1 OPENR, 11, datafile, /F77_UNFORMATTED,/SWAP_IF_LITTLE_ENDIAN
2 READU, 11, ncom
3 READU, 11, un1, vn1, tn1
4 READU, 11, zgeo
5 READU, 11, dichtecgs
6 READU, 11, wn1
7 CLOSE, 11
```

enthalten. Das Schlüsselwort „`/F77_UNFORMATTED`“ teilt der „`OPENR`“-Prozedur mit, dass die Daten unformatiert vorliegen. „`/SWAP_IF_LITTLE_ENDIAN`“ bewirkt, dass die LIMA-Daten, die im Big Endian Format vorliegen, auch so ausgelesen werden wenn auf sie bspw. von einem Windows Betriebssystem zugegriffen wird, das seine Daten im Little Endian Format liest und schreibt. Der Wert „`11`“ ist die Dateieinheit und identifiziert die geöffnete Datei eindeutig, auch im Falle mehrerer geöffneter Dateien. Sie muss bei jedem Lesezugriff angegeben werden. In den Zeilen zwei bis sechs werden die verschiedenen Felder der LIMA-Ergebnisdateien ausgelesen. In Zeile sieben wird der Lesezugriff, mit Angabe der Dateieinheit, beendet. Es müssen immer alle Felder und Werte ausgelesen werden, da das Programm sonst mit einer Fehlermeldung abgebrochen wird.

Die benötigten Winddaten für die Darstellung der Strömungsvisualisierung sind in den Feldern „`un1`“ und „`vn1`“ enthalten. Da sie die Winddaten für beide Hemisphären auf allen 118 Höhenniveaus des LIMA-Modells enthalten, werden sie mittels der „`REFORM`“ Funktion, wie im Quellcodeauszug 4.13, zuerst in den Zeilen eins und zwei auf die gewählte Höhe reduziert und in den Zeilen vier und fünf die Daten auf die Hemisphären aufgeteilt. Anschließend wird das Bild ausgelesen.

Listing 4.13: Extraktion der benötigten Winddaten.

```
1 feld_un1 = REFORM (un1(*,kakt))
2 feld_vn1 = REFORM (vn1(*,kakt))
3
4 feld_un1 = REFORM (feld_un1,igitnb_v2,2)
5 feld_vn1 = REFORM (feld_vn1,igitnb_v2,2)
```

Für die Interpolation von Zwischenwerten der Daten und Bilder werden immer jeweils zwei Datenfelder bzw. Bilder benötigt. Dafür werden immer zwei Felder für die jeweiligen Größen angelegt, eins für vor den interpolierten Zeitschritten und eins für danach. Die innere Schleife, in der die Daten und Bilder interpoliert werden, wird erst ab dem zweiten Schleifendurchlauf der äußeren Schleife aktiviert, wenn jeweils die Vor- und Nacherdaten und Bilder für die Interpolationsschritte vorliegen. Die aktuell ausgelesenen Daten und Bilder werden stets vor dem nächsten Auslesen in die Felder für den vorhergehenden Zeitschritt geschrieben.

Die Anzahl der Schleifendurchläufe der inneren Schleife entspricht den gewünschten Interpolationsschritten zwischen den sechs Stunden-Zeitschritten der Limadaten. Die „`view`"- und „`model`"-Objekte, die in Quellcodeausschnitt 4.7, zum Anfang des Hauptprogramms erstellt worden sind, werden im Prinzip nur als Platzhalter für die Objekttypen in der Übergabestruktur verwendet. Direkt nach dem Beginn der inneren Schleife werden die „`view`"- und „`model`"-Objekte erneut mit denselben Namen erstellt und die alten damit überschrieben. Anschließend werden sie in der Übergabestruktur referenziert. Dieser Schritt ist notwendig, um alle Objekte und damit auch den Hauptspeicher, den sie belegen, nach dem Speichern der Visualisierung in einer Grafikdatei, wieder freigeben zu können. Das geschieht durch die Zerstörung der Objekte am Ende der inneren Schleife. So steht der Hauptspeicher für die nächste Visualisierung wieder zur Verfügung. Dadurch können die Objekte beim nächsten Durchlauf, am Anfang der inneren Schleife, wieder erstellt werden, ohne einen drastischen Performanceverlust eingehen zu müssen.

Als nächstes werden in der Schleife die Bilder, dem aktuell ermittelten Interpolationsintervall entsprechend interpoliert. Die Pixelfarbwerte der eingelesenen Grafiken werden in den Feldern „`background`" und „`background2`" gespeichert. Aus ihnen wird das gewünschte Zwischenbild interpoliert und in das Feld „`rgb`" geschrieben. Die Bildbeschriftung in den Temperaturplots, Tageszeit usw., ändert sich durch die Visualisierung in der Originalzeitauflösung der LIMA-Daten alle sechs Stunden. Um einen Übergangseffekt der Plotbeschriftungen in den interpolierten Bildern zu verhindern, muss die Plotbeschriftung für den Interpolationszeitraum konstant bleiben. Das wird erreicht, indem alle Pixel mit schwarzen und weißen Farbwerten, also Bildhintergrund, Plotbeschriftungen, Kontinentendarstellungen usw., aus dem ersten Bild gefiltert werden, wie es in Zeile eins des Quellcodeauszugs 4.14 zu sehen ist. In Zeile zwei

Listing 4.14: Erhalten der Plotbeschriftung.

```
1  index = WHERE ((background LT 2),count)
2  if count GT 0 THEN rgb[index] = background[index]
3           .
4           .
5           .
6  image = obj_new('IDLgrImage', rgb)
7  model_1->add, image
8  cpalette = obj_new('IDLgrPalette',r,g,b)
9
10 image->setProperty, xcoord_conv=xc, ycoord_conv=yc, $
11          LOCATION=[0,0],PALETTE=cpalette
```

werden die Pixelfarbwerte des zwischen-interpolierten Bilds in dem Feld „`rgb`“, mit den Positionen, die bei der Filterung ermittelt wurden, überschrieben. Diese Methode ist nur anwendbar, wenn die Plotbeschriftungen und der Bildhintergrund Farbwerte besitzen, die nicht in den visualisierten Daten vorkommen. Außerdem dürfen keine anderen grafischen Objekte mit denselben Farbwerten in der Visualisierung vorhanden sein, die sich zeitlich unabhängig von den Plotbeschriftungen ändern. In der Zeile sechs wird das interpolierte Bild als „`image`“-Objekt erstellt und in der darauf folgenden Zeile dem „`model_1`“ hinzugefügt. Die Zeile acht zeigt, dass aus den beim Einlesen erstellten Farbvektoren ein Farbpaletten-Objekt erstellt wird. Das Objekt wird benötigt, damit das „`image`“-Objekt das Bild mit denselben Farbwerten wiedergibt, wie es in den Originalbildern der Fall ist. In den Zeilen `10` und `11` werden die Eigenschaften Position, Skalierung und die Farbpalette des „`image`“-Objekts definiert.

Im anschließenden Programmteil werden die Winddatenfelder linear interpoliert, der Übergabestruktur übergeben und die Unterprogramme „`nachbar`“ und „`Streamline`“ aufgerufen, wie es im Quellcodeauszug 4.15 gezeigt wird.

Nachdem die Strömungslinien in das „`model_2`“ der „`wind_struktur`“ gezeichnet wurden und der Ablauf des Programms in das Hauptprogramm zurückgekehrt ist, wird zunächst das „`model_2`“-Objekt so ausgerichtet, dass die Strömungsvisualisierung mit dem Temperaturplot deckungsgleich abgebildet werden. Darauf folgend findet eine Skalierung mittels der „`Scale`“-Funktion und Ausrichtung mit der

Listing 4.15: Interpolation der Daten und Aufruf der UPs.

```
intpol_1 = 1. / zeitschritt_i * i
intpol_2 = 1. - intpol_1

feld_vn_ip = feld_vn2 * intpol_2 + feld_vn1 * intpol_1
feld_un_ip = feld_un1 * intpol_2 + feld_un2 * intpol_1

;ausgelesene Windwerte der Struktur uebergeben
wind_struktur.u_feld_n = feld_un_ip[*,mhemisphere]
wind_struktur.v_feld_n = feld_vn_ip[*,mhemisphere]

;naechste Nachbarn der letzten Srömungslinienpunkte ermitteln
nachbar,wind_struktur
;Stroemungslinien zeichnen
Streamline,wind_struktur
```

„`Translate`“-Funktion, der „`model`“-Objekte gegenüber dem „`view`“-Objekt statt, sodass die gewünschte Ansicht der kombinierten Visualisierung im Bildbereich zu sehen ist. Durch die Anwendung dieser beiden Funktionen ist es möglich, nicht nur den gesamten Plot wie in Abbildung 4.16 darzustellen, sondern auch gewünschte Ausschnitte vergrößert darzustellen, wie es in der Abbildung 4.17 gezeigt wird.

Nachdem die Ansicht aller grafischen Objekte definiert ist, wird sie mit der Funktion „`draw`“ gerendert, wie in Zeile zwei des Quellcodeausschnitts 4.16 gezeigt wird. Die bisherige mathematische Beschreibung der grafischen Elemente wird damit in einer Bitmap, des „`buffer`“-Objekts festgehalten. Mit „`GetProperty`“ werden die Bilddaten, also die Bitmap in das Feld „`img`“ geschrieben.

Anschließend werden die Speicherpfade und Dateinamen für die Pixelgrafiken generiert. Ein Beispieldateiname ist „`L.NPS.t.c.002.20070118.02_45.png`“. Für

Listing 4.16: Rendern des Bildes.

```
1 buff->draw, wind_struktur.view
2 buff->GetProperty,IMAGE_DATA = img
```

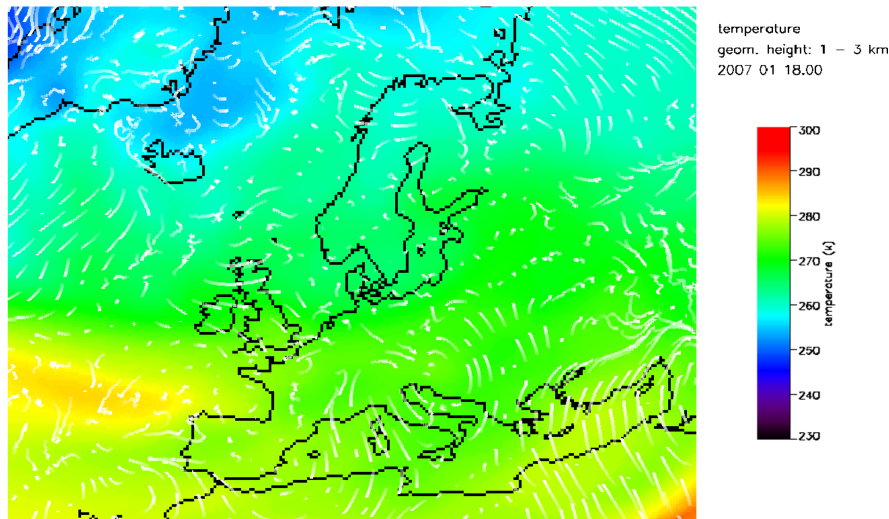

Abbildung 4.17.: Vergrößerter Ausschnitt einer Strömungsvisualisierung, in dem Europa zentriert ist.

eine eindeutige Identifizierung, werden in ihm zwischen den Punkten, von vorne nach hinten, folgende Merkmale bezeichnet:

- „`L`“ Kennzeichnung für eine LIMA-Datenvisualisierung
- „`NPS`“ oder „`SPS`“ für die Hemisphärenkennzeichnung
- „`t`“ visualisierte meteorologische Größe im Hintergrund, hier Temperatur
- „`c`“ Visualisierungsart der hinterlegten Daten, hier Konturplot
- „`002`“ Modellhöhenniveau der visualisierten Größen
- „`20070118`“ Datum, `jjjjmmdd`
- „`02_45`“ Uhrzeit, `hh_mm`

Die Bitmap im Feld „`img`“ wird daraufhin mit der „`Write_PNG`“ Prozedur im PNG-Bildformat gespeichert. Im darauffolgenden Programmabschnitt werden die laufenden Datums- und Zeitangaben inkrementieret und ihr Zustand anschließend abgefragt. Ergeben die Zustandsabfragen, dass der letzte Zeitschritt in der gesetzten Datumsgrenze erreicht ist, wird das Ende des Programms eingeleitet. Ist das nicht der Fall, werden die äußere und innere Schleife erneut durchlaufen. Am Ende der inneren Schleife werden alle „`model`“-Objekte zerstört, um den belegten Hauptspeicher

wieder freizugeben. In den folgenden beiden Abschnitten wird die Umsetzung der Unterprogramme näher beschrieben.

4.5.3. UP-„Nachbar“

Im Unterprogramm „nachbar“ werden die nächsten Gitterpunktnachbarn der aktuell zuletzt ermittelten Strömungslinienendpunkte bestimmt. Aus den umgebenden Nachbarn werden anschließend die Windwerte des Strömungslinienpunkts errechnet und in der Übergabestruktur gespeichert, wie im Funktionsschema in Abbildung 4.18 gezeigt wird.

Zuallererst werden die benötigten Variablen, Felder und Konstanten deklariert. Als nächstes werden die relevanten Daten aus der Übergabestruktur ausgelesen. Im Anschluss beginnt eine Schleife, die das gesamte folgende Unterprogramm umfasst und so oft durchlaufen wird, wie Strömungslinienursprungspunkte definiert sind. Je nach Hemisphärenansicht wird zunächst abgefragt ob der zuletzt ermittelte Strömungslinienendpunkt den Äquator zur gegenüberliegenden Hemisphäre überschritten hat. Ist das der Fall, werden die aktuellen Koordinaten des Strömungslinienpunkts auf die der Ursprungskoordinaten zurückgesetzt. Außerdem wird abgefragt, ob die Strömungslinie ihre definierte maximale Lebenszyklusdauer erreicht hat. Auch in diesem Fall werden die aktuellen Koordinaten des Strömungslinienpunkts zurückgesetzt.

Die Punkte des hier verwendeten LIMA-Modellgitters mit 100 km Kantenlänge liegen auf jeweils festen Breitenringen, wie es auch in Abbildung 4.1 des LIMA-Modellgitters mit 270 km Kantenlänge veranschaulicht ist. Die Breitenringe besitzen einen Abstand von einem Grad zueinander. Daraus ergibt sich, dass für die Beschreibung beider Hemisphären 180 Breitenringe verfügbar sind. Die Breitengradringe liegen immer auf halben Breitengraden, beginnend bei 89,5° bis -89,5°.

In einer Schleife, die alle 90 Breitengrade der gewählten Hemisphäre durchläuft, wird untersucht, zu welchem Breitenringen der Abstand des aktuellen Strömungslinienpunkts kleiner einem Grad ist. Sind die Breitenringe ermittelt, werden die Abstände der Gitterpunkte zueinander auf dem Breitenring in Längengrad bestimmt. In einer weiteren Schleife werden anschließend die Punkte des Breitenringes ermittelt, die dem Strömungslinienendpunkt am nächsten sind. Die Gitterpunkte, deren Abstand

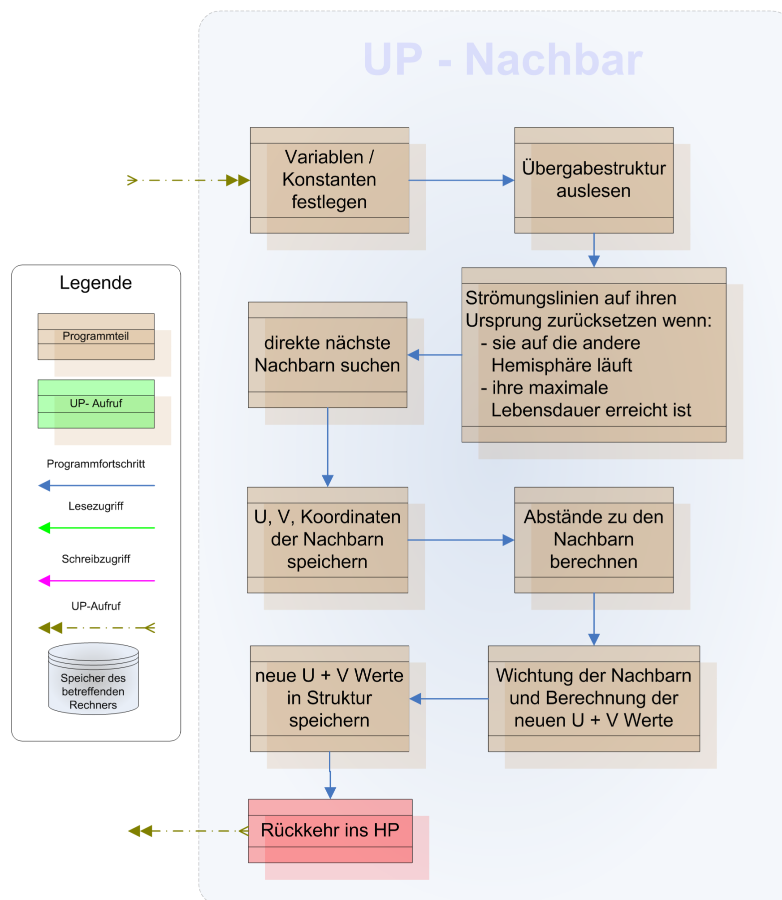

Abbildung 4.18.: Das Bild zeigt das Funktionsschema des Unterprogramms „`Nachbar`“ der Strömungsvisualisierung.

in Längengrad zum Strömungslinienendpunkt kleiner ist als der Abstand der Gitterpunkte auf dem Breitenring zueinander, werden als direkte Nachbarn bestimmt. Dabei wird auch das Randproblem bei einer Nachbarbestimmung links und rechts des Nullmeridians beachtet, der die östliche und westliche Feldgrenze des Gitters darstellt. Ist ein Gitterpunkt als Nachbar ermittelt, werden seine Koordinaten und Werte der u- und v-Windkomponenten in Feldern festgehalten. Wenn die Schleifen durchlaufen wurden, sind vier Nachbargitterpunkte ermittelt.

Nach der Nachbarbestimmung werden die konkreten Abstände der Gitterpunkte in RAD zum Strömungslinienpunkt ermittelt, wie es in Quellcodeauszug 4.17 gezeigt

Listing 4.17: Berechnung der Nachbarabstände.

```
nachbarabstand[n]=(( SIN(phiwind)*SIN(n_phi[n]))+ $
         (COS(phiwind)*COS(n_phi[n])*COS(delta_lamda)))
nachbarabstand[n] = ACOS(nachbarabstand[n])
```

wird. Die Ermittlung basiert auf der Berechnung einer orthodromen Entfernung zweier Punkte. Dabei stellt „`phiwind`“ die Breitengradposition des Strömungsendpunktes, „`n_phi[n]`“ die Breitengradposition des jeweiligen Nachbargitterpunktes und „`delta_lambda`“ den Differenzlängengradwinkel beider Punkte dar. Anschließend wird abgefragt, ob der aktuell betrachtete Gitterpunkt annähernd deckungsgleich auf dem Strömungslinienpunkt liegt. Trifft dies zu, werden die Windwerte des Gitterpunktes vom Strömungslinienpunkt übernommen und ein Indikator „`1`“ gesetzt, um die Punktüberlagerung später abfragen zu können. Die Bedingung für eine Punktüberlagerung ist erfüllt, wenn die Punktpositionen weniger als 0,0001 Rad im Einheitskreis voneinander abweichen, das entspricht auf der Erdoberfläche etwa einer Entfernung von 637 m.

Liegt keine Punktüberlagerung vor, findet eine Wichtung der Nachbarn über ihren Abstand zum Strömungslinienendpunkt statt. Die Wichtung wird in diesem Fall mit dem Prinzip des *gewichteten arithmetischen Mittels* realisiert. Genauer gesagt findet anhand der arithmetischen Mittelung eine räumliche Interpolation statt. Dabei stellen die gewählten Nachbargitterpunkte die Stützstellen für die Interpolation dar. Als Gewichte für das *gewichtete arithmetische Mittel* werden die inversen Distanzen der gewählten Nachbargitterpunkte zum Strömungslinienendpunkt verwendet. So gilt, je näher ein Nachbargitterpunkt dem Strömungslinienendpunkt liegt, desto größer ist sein Einfluss (aus [30]). Die Berechnung der gewichteten Werte wird anhand der folgenden Formel durchgeführt. In der Formel steht g für den jeweiligen u- oder v-Windwert, der interpoliert wird und r für die Streckenentfernungen, wie in Abbildung 4.19 verdeutlicht wird.

$$g_P = \frac{\sum_{i=1}^{n} \frac{g_i}{r_i^2}}{\sum_{i=1}^{n} \frac{1}{r_i^2}} = \frac{g_1\left(\frac{1}{r_1^2}\right) + g_2\left(\frac{1}{r_2^2}\right) + g_3\left(\frac{1}{r_3^2}\right) + g_4\left(\frac{1}{r_4^2}\right)}{\frac{1}{r_1^2} + \frac{1}{r_2^2} + \frac{1}{r_3^2} + \frac{1}{r_4^2}}$$

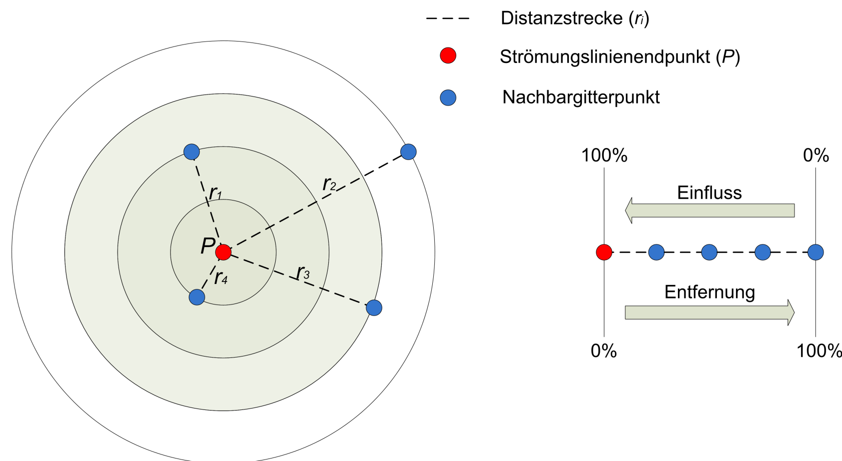

Abbildung 4.19.: Das Bild verdeutlicht das Verfahren der *gewichteten arithmetischen Mittelung*, das zur Werteinterpolation zwischen dem Strömungslinienendpunkt und seiner direkten Gitterpunktnachbarn eingesetzt wird.

Im Quellcodeauszug 4.18 wird die programmiertechnische Umsetzung der *gewichteten arithmetischen Mittelung* gezeigt. In einer Schleife werden die Ergebnisse der einzelnen Terme für jeden Nachbargitterpunkt in den entsprechenden Variablen aufsummiert. Für die v-Windwerte ist das z.B. die Variable „`gewichtwert_v_1`“ für alle Terme über dem Bruchstrich und „`gewichtwert_v_2`“ für alle unter dem Bruchstrich in der Formel. Anschließend werden die aufsummierten Werte der Formel entsprechend verrechnet und die daraus resultierenden Windwerte für den Strömungslinienendpunkt in „`pu`“ bzw. „`pv`“ festgehalten. Diese werden der Übergabestruktur für den entsprechenden Strömungslinienendpunkt übergeben. Nachdem alle Strömungslinienendpunkte abgearbeitet wurden, kehrt das Programm in das Hauptprogramm zurück, indem dann das Unterprogramm aufgerufen wird, das die Strömungslinien zeichnet.

Listing 4.18: Wichtung der Nachbarwerte.

```
FOR l = 0,nachbar_count-1 Do BEGIN

gewichtwert_u_1=gewichtwert_u_1+(n_u[l]* $
    (1/(nachbarabstand[l]*nachbarabstand[l])))

gewichtwert_u_2=gewichtwert_u_2+((1/ $
    (nachbarabstand[l]*nachbarabstand[l])))

gewichtwert_v_1=gewichtwert_v_1+(n_v[l]* $
    (1/(nachbarabstand[l]*nachbarabstand[l])))

gewichtwert_v_2=gewichtwert_v_2+((1/ $
    (nachbarabstand[l]*nachbarabstand[l])))

ENDFOR

pu =(gewichtwert_u_1/gewichtwert_u_2)
pv =(gewichtwert_v_1/gewichtwert_v_2)
```

4.5.4. UP-„`Streamline`“

Im Unterprogramm „`Streamline`“ werden die Strömungslinien realisiert, wie es das Funktionsschema in Abbildung 4.20 verdeutlicht. Dazu werden zunächst die benötigten Variablen und Konstanten festgelegt und anschließend die relevanten Daten aus der Übergabestruktur ausgelesen. Darauf folgend wird die Windrichtung und Windgeschwindigkeit für die aktuellen Strömungslinienendpunkte berechnet. Aus diesen Daten werden die Streckenabschnitte für den vorgegebenen Zeitschritt generiert und daraus wiederum die Zielkoordinaten der Streckenabschnitte ermittelt. Wenn alle benötigten Daten vorliegen, wird die Übergabestruktur mit ihnen aktualisiert und die Strömungslinien gezeichnet.

Im Detail beginnt nach dem Auslesen der Übergabestruktur eine Schleife, die für die Anzahl aller definierten Strömungslinien je einmal durchlaufen wird. Zu Beginn

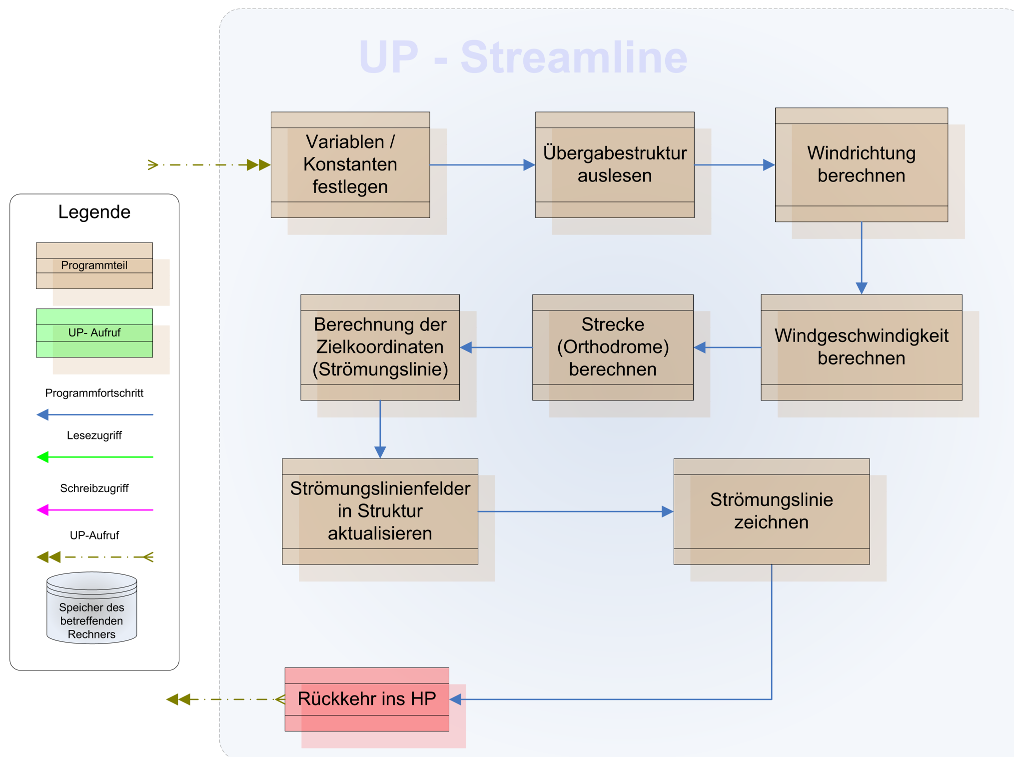

Abbildung 4.20.: Das Bild zeigt das Funktionsschema des Unterprogramms „`Streamline`“ der Strömungsvisualisierung.

der Schleife werden die u- oder v-Windwerte des aktuell behandelten Strömungslinienendpunktes ermittelt. Mit diesen Werten wird anschließend, wie im Quellcodeauszug 4.19 gezeigt wird, die Windrichtung nach der Methode aus Kapitel 4.3.1 ermittelt. Danach wird in Zeile 12 die Windgeschwindigkeit in Meter je Sekunde errechnet. Der Winkel welcher die Windrichtung definiert wird in der Variablen „`alpha`“ festgehalten. Er entspricht dem Winkel α des Punktes A in der Abbildung 4.21. Ist der Winkel der Windrichtung null Grad, dann zeigt der Windvektor entlang des Meridians, der durch den Punkt A verläuft, in Richtung Norden. Der Windvektor dreht sich mit zunehmender Winkelgröße im Uhrzeigersinn (s. Abb. 4.6), d.h. bei einem Winkel von 90 Grad zeigt der Windvektor genau nach Osten.

Anhand der zu diesem Zeitpunkt vorliegenden Daten, wird die Berechnung der Orthodrome auf der Grundlage eines sphärischen Dreiecks im folgenden Programmabschnitt vorgenommen. In der Abbildung 4.6 ist der aktuelle Strömungslinienendpunkt, also der Punkt, von dem aus die Orthodrome berechnet werden soll, der

Listing 4.19: Windrichtung und -Geschwindigkeit ermitteln.

```
if u gt 0. then begin
        alpha1 = (tan(v/u))
        alpha1 =(180. / !dpi) * alpha1
        alpha = 90. - alpha1
endif
if u lt 0. then begin
        alpha2 = (tan(v/u))
        alpha2 =(180. / !dpi) * alpha2
        alpha = 270. - alpha2
endif
        ;Windgeschwindigkeit in m/s
        wind_v= (sqrt((u*u)+(v*v)))
```

Punkt A. Der Winkel der Windrichtung ist in α enthalten. Die Strecke $c_strecke$ stellt die gesuchte Orthodrome und B ihren gesuchten Endpunkt dar. Für die Berechnung des Endpunktes B wird die Länge der Orthodrome als Zentriwinkel ω, d.h. den zum Bogen der Orthodrome gehörenden Mittelpunkswinkel benötigt. Der Mittelpunkt bezieht sich in diesem Fall auf den der Kugel, der in der Abbildung 4.21 mit *0* gekennzeichnet ist. Dazu wird zunächst die Strecke der Orthodrome in Metern ermittelt, die sich aus Windgeschwindigkeit in $\frac{m}{s}$ $\times$ Zeitschritt in Sekunden ergibt. Danach wird der Erdumfang in Grad (360°) durch den mittleren Erdumfang in Metern (ca. 40030000 m) geteilt und mit der ermittelten Strecke der Orthodrome in Metern multipliziert. Das Ergebnis wird in der Variablen $c_strecke$ festgehalten. Die Koordinaten des Punktes A sind in den Winkelangaben *Phi* und *Lambda* gegeben. Dabei entspricht *Phi* der Strecke $b_strecke$ vom Punkt A auf seinem Meridian zum Nordpol. Die Strecke zwischen Punkt B und dem Nordpol in Grad, also $a_strecke$ (s. Abb. 4.21), ist jetzt mit den gegebenen Größen mittels des Seitenkosinussatzes berechenbar. Dafür ergibt sich folgende Gleichung:

$$\cos(a_strecke) = \cos(b_strecke)\cos(c_strecke) + \sin(b_strecke)\sin(c_strecke)\cos(\alpha).$$

Die Länge der Strecke *a_strecke* in Grad entspricht dem Winkel *Phi* der Koordinaten des Punktes *B*. Mit der zusätzlich errechneten Größe *a_strecke* wird anschließend mit der folgenden Umformung des Seitenkosinussatzes der Winkel β errechnet.

$$\cos(\beta) = \frac{\cos(b_strecke) - \cos(c_strecke)\cos(a_strecke)}{\sin(c_strecke)\sin(a_strecke)}$$

Der Winkel β ist für die Berechnung der Koordinaten des Punktes *B* nicht zwingend notwendig. Die Größe wird jedoch benötigt, wenn die berechneten Größen des vorliegenden sphärischen Dreiecks auf ihre Richtigkeit überprüft werden sollen. Die

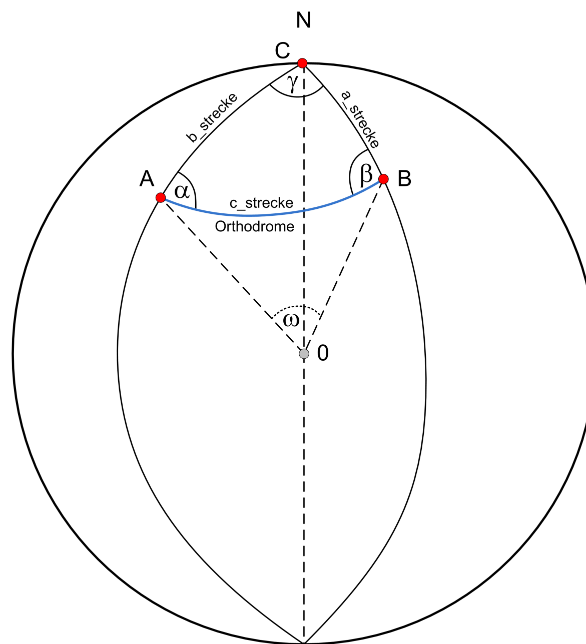

Abbildung 4.21.: Das Bild verdeutlicht die zur Berechnung der Orthodrome verwendeten Punkte, Strecken und Winkel.

Bestimmung des Winkels γ findet durch die folgende umgeformte Seitenkosinussatzgleichung statt.

$$\cos(\gamma) = \frac{\cos(a_strecke)\cos(b_strecke) - \cos(c_strecke)}{\sin(a_strecke)(-1)\sin(b_strecke)}$$

Der Winkel γ stellt die Differenz zwischen den *Lambda* Koordinaten der Punkte A und B dar. Die Berechnung der *Lambda* Koordinate von Punkt B ist abhängig von der Größe des Winkels α. Ist er kleiner 180°, zeigt also in östliche Richtungen, ergibt sich *Lambda* von B durch *Lambda* $A + \gamma$. Ist α größer 180° und zeigt damit in westliche Richtungen, ergibt sich *Lambda* B durch *Lambda* $A - \gamma$. Tritt zwischen den Punkten A und B eine Überquerung des Nullmeridians auf, ist *Lambda* B negativ oder weist Werte größer 360° auf. In so einem Fall werden durch Abfragen im Programm entsprechend 360° addiert oder subtrahiert, damit die Werte wieder in einem gültigen Bereich liegen. Für die Überprüfung auf Korrektheit der berechneten Größen kann auf den trigonometrischen Sinussatz zurückgegriffen werden, der besagt, dass das Verhältnis aller drei Seiten des sphärischen Dreiecks zu ihren Winkeln gleich groß sein muss. Dies kann auch im Programm anhand des Quellcodeabschnitts 4.20 zu Testzwecken, durch entfernen der Kommentarzeichen genutzt werden.

Im folgenden Programmabschnitt werden die neu errechneten Koordinatendaten der Übergabestruktur übergeben. Für die definierte Anzahl der sichtbaren Streckenabschnittspunkte jeder Strömungslinie ist zu Beginn des Programms das Feld „`streamline_field_s`“ deklariert worden. Dort sind die jeweiligen *Phi* und *Lambda*-Werte der Punkte gespeichert. Vor der Speicherung der neuen Koordinaten im Feld werden mit der Funktion „`SHIFT`“ alle Koordinateneinträge der aktuellen Strömungslinie um einen Feldeintrag nach hinten verschoben. Dadurch wird der erste Feldeintrag zum zweiten usw.. Der bisher letzte Feldeintrag geht dabei verloren. Die

Listing 4.20: Sphärische Berechnungen verifizieren.

```
    ;aver = sin(a_strecke) / sin(alpha_bc)
    ;print,'aver: ',aver
    ;bver = sin(b_strecke) / sin(beta_ac)
    ;print,'bver: ',bver
    ;cver = sin(c_strecke) / sin(gamma_ab)
    ;print,'cver: ',cver
```

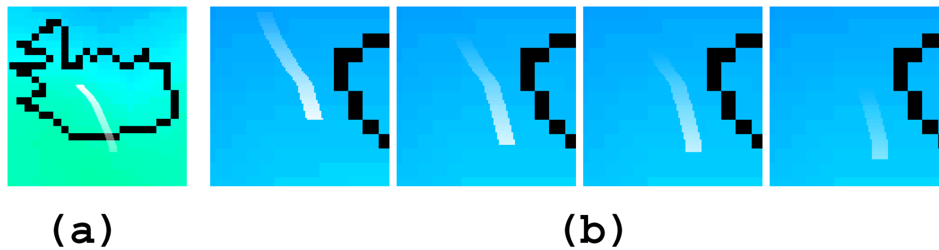

Abbildung 4.22.: Das Bild verdeutlicht stark vergrößert die Transparenzeffekte der Strömungslinien, von ihrer Länge abhängig in (a) und beim Ausblenden ihres Lebensalters abhängig in den Bildern (b).

Strömungslinie soll nur eine definierte Anzahl Streckenabschnitte sichtbar sein. Zum Ende hin nimmt die Transparenz der einzelnen Streckenabschnitte in dem Maße zu, dass die Streckenabschnitte hinter der definierten Anzahl unsichtbar sind und somit auch ihre Koordinaten überflüssig werden (s. Abb. 4.22 (a)). Die ersten Koordinateneinträge der Strömungslinie nehmen dabei den Wert Null an. Diese werden anschließend mit den zuvor errechneten *Phi* und *Lambda*-Werten überschrieben.

Als nächstes wird im laufenden Programm das Alter der Strömungslinie abgefragt. Nähert sie sich ihrer maximal definierten Anzahl an Zeitschritten, wird sie ausgeblendet damit sie am Ende ihres Lebenszykluses nicht abrupt aus der Visualisierung verschwindet (s. Abb. 4.22 (b)). Das wird durch eine Abfrage realisiert, die feststellt, ob die Strömungslinie ihrer maximalen Anzahl an Zeitschritten näher ist als sie selbst lang ist. Trifft das zu, wird ein Transparenzwert errechnet, der abhängig von der Entfernung der Strömungslinie zu ihrem definierten Ende immer mehr zu nimmt. Dieser Wert wird in der Variablen „`alphazusatz`“ festgehalten und später dem Transparenzwert, der für das Ausblenden des Endes der Strömungslinie sorgt abgezogen, sodass die Gesamttransparenz zunimmt. Trifft die Abfrage nicht, zu wird dieser Wert Null gesetzt.

Im darauf folgenden Programmteil wird die Strömungslinie gezeichnet. Das geschieht in einer Schleife, die für jeden Streckenabschnitt der sichtbaren Länge der Strömungslinie je einmal durchlaufen wird. Dafür findet zuerst eine Koordinatentransformation von Polarkoordinaten in kartesische Koordinaten statt, wie es in Quellcodebeispiel 4.21 verkürzt gezeigt wird. Da für das Zeichnen der Polygonlinie ein Anfangs- und ein Endpunkt benötigt wird, werden die Koordinaten des jeweils

Listing 4.21: Koordinatentransformation der Streckenpunkte.

```
1 x_d[0]=r*(cos(phi,i))*(cos(lambda,i))+x_achsenkorrektur
2 x_d[1]=r*(cos(phi,i+1))*(cos(lambda,i+1))+x_achsenkorrektur
3 y_d[0]=r*(cos(phi,i))*(sin(lambda,i))+y_achsenkorrektur
4 y_d[1]=r*(cos(phi,i+1))*(sin(lambda,i+1))+y_achsenkorrektur
5 z_d[0]=r*(sin(phi,i))
6 z_d[1]=r*(sin(phi,i+1))
7 alphawert=(1.-((1./(FLOAT(maxline)))*i))-alphazusatz
```

aktuellen und nachfolgenden Strömungslinienstreckenpunktes berechnet und in den entsprechenden Vektor geschrieben. Den x- und y-Werten wird jeweils ein Korrekturwert addiert, der es ermöglicht, die Strömungsvisualisierung exakt über den Temperaturplot zu positionieren. In Zeile sieben wird der Transparenzwert des zu zeichnenden Streckenabschnitts berechnet. Er ist abhängig von der maximalen Anzahl der sichtbaren Streckenabschnitte in „`maxline`“, dem aktuellen Streckenabschnitt „`i`“ und dem zusätzlichen Transparenzwert „`alphazusatz`“, wie oben bereits beschrieben. Um negative Transparenzwerte zu verhindern, werden entsprechende Werte in einer Abfrage Null gesetzt. Anschließend wird der Streckenabschnitt als ein „`IDLgrPolyline`“-Objekt in den dreidimensionalen Raum des `Model`-Objekts für die Strömungslinien gezeichnet, wie es in Quellcodeausschnitt 4.22 gezeigt wird.

Die Visualisierung der Strömungslinien auf der dreidimensionalen Halbkugel erzeugt bei der polarstereografischen Draufsicht, vom Pol aus gesehen zum Äquator hin eine zunehmende Verzeichnung. Diese Verzeichnung entspricht der des Temperaturplots, der ursprünglich auch auf einer dreidimensionalen Halbkugel visualisiert wurde. Die Methode der Kombination des zweidimensionalen Hintergrundbildes

Listing 4.22: Strömungslinienabschnitt zeichnen.

```
mypolyline_1=OBJ_NEW('IDLgrPolyline',x_d,y_d,z_d, $
    COLOR=weiss, LINESTYLE = 0,Thick=3,       $
    ALPHA_CHANNEL = alphawert)

wind_struktur.model_2->add, mypolyline_1
```

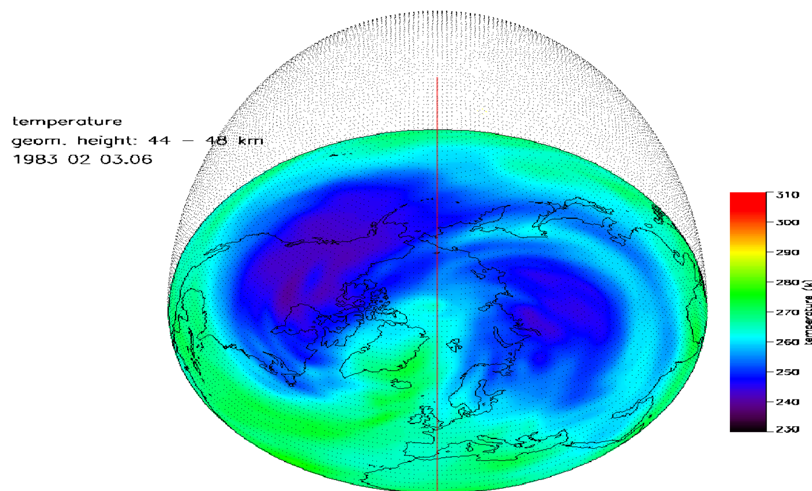

Abbildung 4.23.: Das Bild verdeutlicht die Kombination der dreidimensionalen Halbkugel im Modellobjekt für die Strömungslinien und den darunter positionierten Temperaturplot im entsprechenden Modellobjekt. Beide Modelle sind für die bessere Veranschaulichung um 45° nach hinten gekippt. Der Nullmeridian ist rot eingezeichnet.

mit der dreidimensionalen Halbkugel wird in Abbildung 4.23 veranschaulicht. Für ein besseres Verständnis sind die beiden entsprechenden Modelle auf der x-Achse um 45° nach hinten gedreht und der Nullmeridian rot eingezeichnet.

Nach dem Zeichnen der Strömungslinie wird ein Index für deren Lebensdauer in der Übergabestruktur um eins erhöht. Nachdem alle vorhandenen Strömungslinien gezeichnet sind, wird in das Hauptprogramm zurückgekehrt.

4.5.5. Fazit der Ergebnisvisualisierung

Das entstandene Visualisierungsprogramm stellt Wind- und Temperaturdaten des LIMA-Modells dar (s. Abb. 4.24). Die Darstellungsform ist dabei von den bereits vorliegenden Temperaturvisualisierungen als polarstereografische Projektion vorgegeben. Die Realisierung der angestrebten Strömungslinie, die zu ihrem Ende hin transparent ausklingt, ist wie gezeigt wurde, mittels des *IDL Object Graphics System* möglich. Die vorliegenden Temperaturvisualisierungen sind mittels des *IDL Direct Graphics System* realisiert worden. Die Kombination von grafischen Elementen

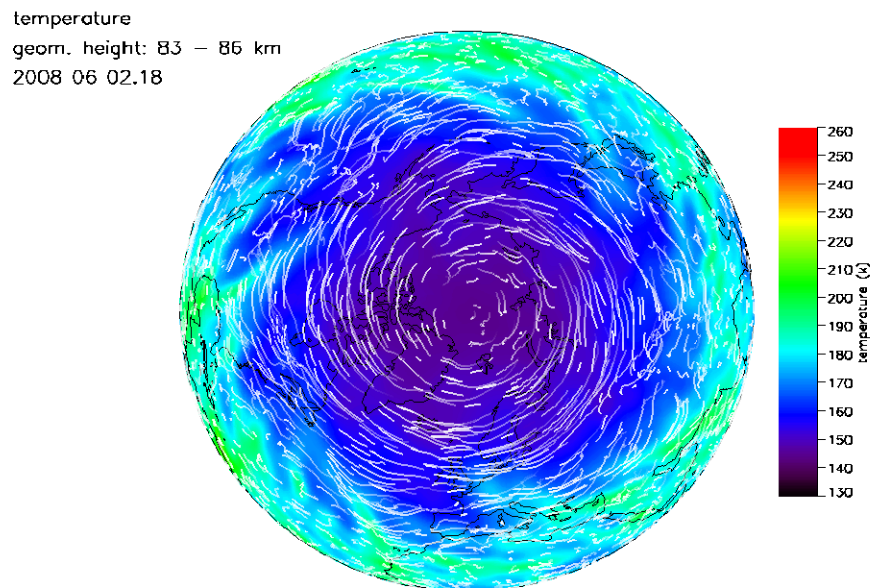

Abbildung 4.24.: Einzelbild der Strömungsvisualisierung.

der beiden IDL- Grafiksysteme ist nicht direkt möglich. Durch die Kombination der vorher erstellten Einzelbilder der Temperaturvisualisierung und der nachträglich eingefügten Strömungsvisualisierung wird die Möglichkeit der indirekten Kombination der beiden IDL Grafiksysteme gezeigt. Die verwendete Methode der Bildinterpolation, die auf die Temperaturvisualisierungen angewendet wird, ergibt annähernd gleiche Ergebnisse wie die Methode, die Daten vor der Visualisierung, dem gewünschtem Zeitintervall angepasst, zu interpolieren. Das trifft allerdings nur auf Flächendarstellungen ähnlich den verwendeten Temperaturvisualisierungen zu, die zwischen den Einzelbildern keine zu großen Änderungen aufweisen. Vorteile der angewendeten Methode sind der eingesparte Programmier- und Rechenaufwand für die erneute Temperaturvisualisierung. Mit ihr ist z.B. auch noch die Erstellung zeitlich hochaufgelöster Animationen möglich, wenn die zu den vorhandenen Einzelbildern gehörenden Originaldaten verloren gegangen sein sollten. Die atmosphärischen Strömungen sind in der erarbeiteten Visualisierung sehr gut erkennbar. Das gilt für Standbilder sowie für Animationen. Bei der Betrachtung der Animationen werden zusätzlich Zusammenhänge zwischen den Bewegungen der Temperaturstrukturen und der atmosphärischen Strömungen erkennbar. Für optimale Strömungsvisualisierungen müssen die Eigen-

schaften der Strömungslinien wie Breite, sichtbare Länge, Lebenszyklusdauer sowie die Gesamtanzahl der Linien individuell angepasst werden. Sie sind abhängig von z.B. der Größe des Ausschnitts der Visualisierung, den Windgeschwindigkeiten im dargestellten Windfeld sowie dem Darstellungsmedium. Im Laufe der Arbeit wurde dazu festgestellt, dass bei Druckmedien die Linien wesentlich dicker gezeichnet werden müssen, um denselben Darstellungseffekt zu erzielen wie bei aktiven Ausgabemedien, z.B. Bildschirm oder Projektor.

Für die korrekte Darstellung der Daten in der Visualisierung sind im Laufe der Programmentwicklung zahlreiche Tests durchgeführt worden. So wurde z.B. ein Programm entwickelt, das die korrekte Arbeitsweise der Algorithmen zur Realisierung der Strömungslinien in einem statischen Windfeld testet. Die Abbildung 4.25 zeigt den Ausschnitt eines Ergebnisbildes dieses Tests. Das Testprogramm bildet den Temperaturplot und die Strömungsdaten in derselben Form ab wie das eigentliche Visualisierungsprogramm. Es werden zusätzlich zur Strömungslinie alle sichtbaren LIMA-Gitterpunkte und die dazugehörigen Windvektoren gezeichnet. Während der Visualisierung der Strömungslinie werden zusätzlich die Nachbargitterpunkte gelb markiert, die zur Interpolation der Windwerte an den Strömungslinienpunkten verwendet werden. Auf diese Art kann visuell nachvollzogen werden, ob sich die Strömungslinie korrekt im Windfeld bewegt.

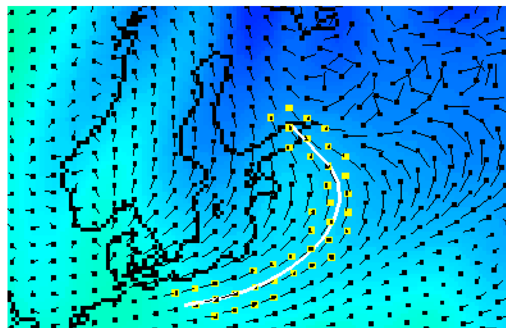

Abbildung 4.25.: Das Bild zeigt das Ergebnis eines Tests der Algorithmen zur Realisierung der Strömungsvisualisierung. Dargestellt sind die Strömungslinie (weiß), die verwendeten nächsten Nachbargitterpunkte (gelb), die Modellgitterpunkte mit den in ihnen enthaltenen Windvektoren(schwarz) und die hinterlegten Temperaturen mit Kontinentenumrandungen.

Die Strömungsvisualisierungen sind parallel zu den vorliegenden Temperaturplots in den LIMA-Modellhöhenniveaus 2 (∼1-3 km), 42 (∼43-48 km) und 75 (∼82-86 km) durchgeführt worden. Die Interpolation der LIMA-Winddaten in der Strömungsvisualisierung wirkt sich erst im Höhenniveau 75, das dem der NLC entspricht, merklich aus. In den darunter liegenden Höhenniveaus ändern sich die Windfelder so geringfügig, dass Visualisierungen in einer zeitlichen Auflösung der LIMA-Daten entsprechend von sechs Stunden eine harmonisch ablaufende Animation der Daten ergibt. Im Höhenniveau 75 ist das nicht mehr der Fall, dort ist die Atmosphäre so dynamisch, dass eine sechs Stunden aufgelöste Animation sehr abgehackt erscheint. Bei der Generierung von stündlichen Bildern der Visualisierung mit den jeweils für sechs Stunden statischen Originalwindfeldern, ohne Interpolation, springen die Strömungslinien förmlich in einem Zick-Zack-Muster, bedingt durch die sehr unterschiedlichen Windfelder (s. Abb. 4.26 (a)). Bei derselben Art der Visualisierung mit zusätzlich entsprechend dem Visualisierungszeitschritt interpolierten Windfeldern verschwinden die Zick-Zack-Muster vollkommen (s. Abb. 4.26 (b)). In der Animation der Einzelbilder wird dadurch eine deutlichere Korrelation der Strömungslinienbewegung mit den darunter liegenden Temperaturmustern sichtbar. Durch die LIMA-Dateninterpolation resultiert in der Modellhöhe 75 also eine Fehlerbeseitigung und ein visueller Informationsgewinn. Der im Abschnitt 4.4.2

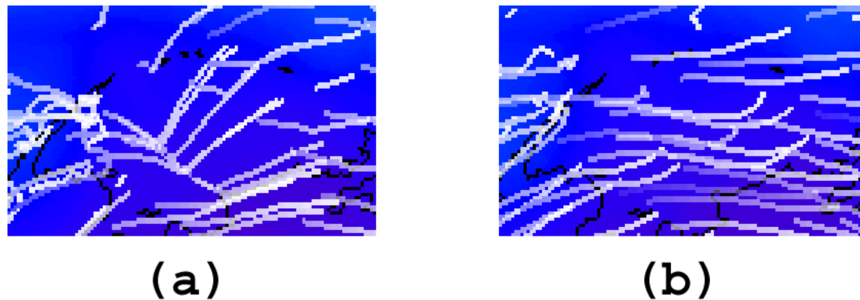

Abbildung 4.26.: Im Bild (a) ist eine Strömungsvisualisierung auf dem LIMA-Modellhöhenniveau 75 mit stündlich erzeugten Bildern, aber für jeweils sechs Stunden statischen Windfeldern gezeigt. Bild (b) zeigt eine Visualisierung zur gleichen Zeit, am gleichen Ort, in der gleichen Höhe und der gleichen stündlichen Auflösung, jedoch mit stündlich interpolierten Windfeldern.

angesprochene Effekt der Polygonlinien, bei dem die Transparenz der Polygone keine Auswirkungen auf überzeichnete Polygone zeigt, hat keine negativen Auswirkungen auf das Gesamtbild der Visualisierungen.

Durch die im Laufe der Entwicklung entstandenen Testmethoden wurde sichergestellt das Expressivitätskriterium der Visualisierung zu erfüllen. Die hohe Effektivität der atmosphärischen Strömungsvisualisierung wurde in den theoretischen Vorbetrachtungen bereits beschrieben und ist auch in der realisierten Version wiederzufinden. Das Angemessenheitskriterium der Visualisierung wurde erfüllt, indem die Möglichkeit genutzt wurde, die Vorteile der vorhandenen IDL-Grafiksysteme zu kombinieren und so der Programmier- und Rechenaufwand möglichst gering gehalten wurde.

Für die Erstellung der Animationen aus den Einzelbildern ist das Programm VideoMach unter Windows verwendet worden. Bei der Ausführung der Visualisierung auf dem Abteilungsserver können vorhandene Skriptroutinen genutzt werden, die unter Verwendung der Programme Convert und MJPEGTOOLS Animationen erstellen.

Bei zukünftigen Strömungsvisualisierungen mit Hintergrunddaten und anderen Projektionsarten wird empfohlen, die in der vorliegenden Arbeit erörterte Methode für die Kombination der verschiedenen Daten anzuwenden, um die jeweiligen Vorteile der IDL-Grafiksysteme zu nutzen. Für die in der vorliegenden Arbeit realisierte Strömungsvisualisierung ist angestrebt, das Programm auf dem Abteilungsserver Photon einzupflegen, sodass das Programm unabhängig von einem bestimmten Arbeitsplatz-PC ausführbar ist. Weiterhin sollten die Möglichkeiten untersucht werden, die Anzahl der Strömungslinien in der Visualisierung zu vergrößern. Das gilt vor allen Dingen bei der Darstellung von vergrößerten Ausschnitten der Visualisierung. Lösungsansätze dafür sind z.B. die Ursprungspunkte der Strömungslinien vom LIMA-Modellgitter zu lösen und so eine von diesem unabhängige Ursprungspunktanzahl zu generieren. Weiterhin könnten von einem Strömungslinienursprungspunkt aus mehrere Strömungslinien in gleichen Zeitabständen hintereinander in das Windfeld starten oder die Anzahl der in das Windfeld startenden Strömungslinien von den örtlichen Windstärken abhängig gemacht machen.

5. Zusammenfassung und Ausblick

Im Rahmen der vorliegenden Master-Thesis ist die Umsetzung eines atmosphärischen Strömungsfilmes auf der Grundlage von Ergebnisdaten des LIMA-Modells mit IDL am IAP-Kühlungsborn untersucht und realisiert worden. Für den Strömungsfilm dient eine Strömungsvisualisierung der Universität Basel als Vorlage, deren Eigenschaften eingehend untersucht wurden. In der Visualisierung werden Temperaturdaten mit darüber gelegten Strömungslinien dargestellt. Schwerpunkt der Strömungsvisualisierung sind die Strömungslinien und deren Eigenschaften. Für die Realisierung der kombinierten Darstellung der meteorologischen Größen Wind und Temperatur sowie die Realisierung der Strömungslinien mit IDL sind umfangreiche Untersuchungen durchgeführt worden. Diese ergaben, dass die Realisierung der Strömungslinien, die am Ende einen zunehmenden Transparenzeffekt aufweisen, ausschließlich mit dem *IDL Object Graphics System* der beiden zur Verfügung stehenden IDL-Grafiksysteme möglich ist. Das daraus resultierende IDL-Programm realisiert die Strömungsvisualisierung, indem schon vorhandene Temperaturvisualisierungen in polarstereografischer Projektion als Hintergrundbilder geladen werden und die eigentliche Strömungsvisualisierung darüber gezeichnet wird. Dabei wird diese in derselben polarstereografischen Projektion der Temperaturvisualisierungen dreidimensional nachgebildet. Da die verwendeten Temperaturvisualisierungen und die Winddaten aus den LIMA-Modellergebnisdaten jeweils eine zeitliche Auflösung von sechs Stunden besitzen, sind Algorithmen implementiert worden, die es ermöglichen, diese Eingabedaten zwischen ihrer zeitlichen Auflösung beliebig oft zu interpolieren. Für die korrekte Darstellung der meteorologischen Größen sind zahlreiche Tests durchgeführt worden. Mit dem aktuell vorliegenden Visualisierungsprogramm ist zusätzlich zur polarstereografischen Projektion auch die Vergrößerung einzelner Ausschnitte möglich. Des Weiteren können sämtliche Eigenschaften der Strömungslinien den verwendeten Winddaten so angepasst werden, dass eine möglichst effektive Strömungsvisualisierung resultiert. Die

endgültige Strömungsvisualisierung wird in Form von Einzelbildern gespeichert, die mit verschiedenen Programmen zu einem Film konvertiert werden können. Das Visualisierungsprogramm, das in der Bearbeitungszeit der vorliegenden Master-Thesis entwickelt wurde, liegt in einer Version vor, welche die zu Beginn der Arbeit definierten Anforderungen erfüllt. Es wird angestrebt das Programm auf dem Abteilungsserver Photon einzupflegen, um seine Kapazitäten für die rechenintensive Strömungsvisualisierung nutzen zu können. Die Art der Möglichkeit zur Umsetzung der Strömungsvisualisierung mit IDL kann auch auf andere Modellergebnisse des IAP angewendet werden.

Zusätzlich ist im Rahmen dieser Arbeit eine ausführliche Beschreibung der Netzwerk- und Großrechnerumgebung mit praktischen Anwendungsbeispielen entstanden, die den Mitarbeitern des IAP zukünftig als Einführungsmaterial bzw. Nachschlagewerk zur Verfügung steht. Sie enthält Hinweise und Tipps zur Verwendung der Großrechentechnik am IAP und gibt eine Einführung in die parallele Programmierung mit Fortran und OpenMP. Um die Anwendungsgebiete der einzelnen Großrechner zu verdeutlichen, wurden unterschiedliche Benchmarks durchgeführt und deren Ergebnisse ausführlich ausgewertet. Das entstandene Benutzerhandbuch sollte kontinuierlich den Änderungen der Netzwerk- und Großrechnerumgebung angepasst werden.

6. Danksagung

Ich danke Herrn Prof. Dr. Lübken für die Aufnahme in das Institut. Herrn Dr. Berger und Herrn Dr. Baumgarten danke ich für die Aufnahme in die Arbeitsgruppe und für ihre ständige Diskussionsbereitschaft sowie die vielen wertvollen Anregungen und Hinweise bei der Erstellung dieser Arbeit. Außerdem möchte ich mich noch bei allen nicht namentlich genannten Mitarbeitern des Instituts bedanken, die mich durch ihr freundliches Entgegenkommen und ihrer Hilfe bei dieser Arbeit unterstützt haben.

Literaturverzeichnis

[1] D. Rachholz. *Untersuchungen zur automatisierten Filmerstellung einer Eiswolke.* Hochschule Wismar, IAP-Kühlungsborn, 2005.

[2] D. Rachholz. *Entwicklung einer automatisierten Software zur Darstellung des Inhalts einer meteorologischen Datenbank.* Hochschule Wismar, IAP-Kühlungsborn, 2006.

[3] Prof. Dr. F. J. Lübken, Prof. Dr. G. Schmitz, and Dr. J. Bremer. *Institutsbericht 2006/2007, IAP-Kühlungsborn.* IAP-Kühlungsborn, 2007.

[4] G. Baumgarten, M. Gerding, B. Kaifler, and N. Müller. *A trans-european network of cameras for observation of noctilucent clouds from 37°N to 69°N.* IAP-Kühlungsborn, 2009.

[5] Rainer Krienke. *Kommunikation unter Linux.* SuSE Press, SuSE Linux AG, 2., aufl. edition, 2003.

[6] *Handbuch der Netzwerktechnologien.* Cisco Press, Markt und Technik, 2001.

[7] Dr. Andreas Voss. *Das große PC und Internet Lexikon.* Data Becker Verlag, 2., aufl. edition, 2003.

[8] Yves Lepage and Paul Iarrera. *Die Unix Sysad Bibel.* MITP-Verlag, 1., aufl. edition, 1999.

[9] Reiner Vogelsang. *SGI® Altix^{TM} Hardware Architektur.* SGI GmbH, 2006.

[10] Shameem Akhter and Jason Roberts. *Multicore Programmierung.* Intel Press, 1., aufl. edition, 2008.

[11] PD Dr. Alfred Strey. *High Performance Computing.* Universität Ulm, (http://www.informatik.uni-ulm.de/ni/Lehre/SS04/HPC/HPCcluster2.pdf), 2004.

[12] *Intel® Fortran Compiler 10.0 for Linux Release Notes.* University of Illinois, (http://www.ncsa.illinois.edu/UserInfo/Resources/Software/Intel/Compilers/10.0/F_Release_Notes.htm), 2009.

[13] *OpenMP Environment Variables.* Intel® (http://www.intel.com/software/products/compilers/docs/flin/main_for/mergedprojects/optaps_for/common/optaps_par_var.htm), 2009.

[14] *Intel® Compiler support for OpenMP libraries.* Intel® (http://www.intel.com/software/products/compilers/docs/flin/main_for/mergedprojects/optaps_for/common/optaps_par_libs.htm), 2009.

[15] B. Chapman, G. Jost, and R. van de Pas. *Using OpenMP - Portable Shared Memory Parallel Programming.* MIT Press, 2008.

[16] B. Chapman, G. Jost, and R. van de Pas. *Using OpenMP-Portable Shared Memory Parallel Programming.* MIT Press, 1., aufl. edition, 2008.

[17] R. Chandra, L. Dagum, D. Kohr, D. Maydan, et al. *Parallel Programming in OpenMP.* ACADEMIC PRESS, 2001.

[18] Hägar Holste. *Aufbau einer tageslichtfähigen Nachweisbank für ein Rayleigh-Lidar.* IAP-Kühlungsborn, 2007.

[19] Olga Alicja Suminska. *temperature anemometer for balloon-borne stratospheric turbulence soundings.* IAP-Kühlungsborn, 2008.

[20] M. Bender and M. Brill. *Computergrafik - Ein anwendungsorientiertes Lehrbuch.* Carl Hanser Verlag, 2., aufl. edition, 2005.

[21] H. Schumann and W. Müller. *Visualisierung - Grundlagen und allgemeine Methoden.* Springer Verlag, 1., aufl. edition, 2000.

[22] Universität Basel. *Meteoblue - Numerical Weather Prediction.* (http://www.unibas.ch/geo/mcr/3d/meteo/index.dt.htm), 2006.

[23] McSush. *Orthodrome globe.* Wikimedia Commons, (http://commons.wikimedia.org/wiki/File:Orthodrome_globe.svg), 2009.

[24] *IDL Making Innovation Easier.* ITT Visual Information Solutions, (http://www.ittvis.com/ProductServices/IDL/LatestRelease.aspx), 2009.

[25] Marlene Busch, Reimar Bauer, Heinz Heer, and Michael Wagener. *Praxisbezogene IDL Programmierung.* Forschungszentrum Jülich GmbH, 2., aufl. edition, 2008.

[26] Liam e. Gumley. *Practical IDL Programming.* Morgan Kaufmann, 1., aufl. edition, 2001.

[27] Kenneth P. Bowman. *An Introuduction to Programming with IDL.* Academic Press, 1., aufl. edition, 2005.

[28] Ronn Kling. *Power Graphics with IDL: A Beginners Guide to IDL Object Graphics.* KRS, 1., aufl. edition, 2002.

[29] D.Fanning. *Coyote's Guide to IDL Programming.* Fanning Consulting, 2009.

[30] Martin Herold. *HS Analyse und Modellierung räumlicher Daten.* Friedrich-Schiller-Universität Jena, 2004.

Abkürzungsverzeichnis

ALU	**A**rithmetic **L**ogic **U**nit
ccNUMA	**c**ache **c**oherent **N**on-**U**niform **M**emory **A**ccess
CD	**C**ompact **D**isc
CPU	**C**entral **P**rocessing **U**nit
DMF	**D**ata **M**igration **F**ilesystem
DMZ	**D**e**M**ilitarisierte **Z**one
ECMWF	**E**uropean **C**enter for **M**edium range **W**eather **F**orecast
gcc	**g**nu **c**ompiler **c**ollection
gdb	**g**nu **de**bugger
GCM	**G**lobal **C**irculation **M**odel
GNU	rekursives Akronym von **G**NU is **N**ot **U**nix (GNU-Projekt)
IA-64	**I**ntel **A**rchitecture **64**-Bit (64-Bit-Architektur und Befehlssatz von Intel für die Prozessorgenerationen Itanium und Itanium 2)
IAP	Leibniz-**I**nstitut für **A**tmosphären**P**hysik e.V. Kühlungsborn
idb	**i**ntel **deb**ugger
ifort	**i**ntel **fort**ran Übersetzer
IDL	**I**nteractive **D**ata **L**anguage (Research Systems, Inc.)
IMKL	**I**ntel **M**athematic **K**ernel **L**ibrary
ipp	Intel **i**ntegrated **p**erformance **p**rimitives (Bibliothek)
LAN	**L**ocal **A**rea **N**etwork
LIMA	**L**eibniz-**I**nstitut **M**iddle **A**tmosphere Model
LTO	**L**inear **T**ape **O**pen
MPI	**M**essage **P**assing **I**terface
NAG	**N**umerical **A**lgorithms **G**roup (Bibliothek)
NLC	**N**octi**L**ucent **C**loud (leuchtende Nachtwolke)
NPS	**N**ord**P**olar**S**tereografisch (Projektionsart)
NUMA	**N**on-**U**niform **M**emory **A**ccess
openMP	**open** specifications for **M**ulti-**P**rocessing
PC	**P**ersonal **C**omputer
PMSE	**P**olar **M**esosphere **S**ummer **E**cho
PMWE	**P**olar **M**esosphere **W**inter **E**cho

PNG	**P**ortable **N**etwork **G**raphics
RAM	**R**andom **A**ccess **M**emory
RAR	**R**oshal **AR**chive
SPS	**S**üd**P**olar**S**tereografisch (Projektionsart)
UP	**U**nter**P**rogramm
VPN	**V**irtual **P**rivate **N**etwork
WAN	**W**ide **A**rea **N**etwork

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der aufgeführten Hilfsmittel angefertigt habe.

Ort, Datum   Unterschrift

A. Anhang

- Thesen
- CD mit IDL Programm, Dokumentation und Ergebnissen

A. Thesen

Master-Thesis
Animation von multidimensionalen atmosphärischen Parametern zur Untersuchung von leuchtenden Nachtwolken

Eingereicht am:	14. Januar 2010
von:	Dirk Rachholz geb. am 28.09.1979 in Kühlungsborn
Betreuer, Einrichtung:	Prof. Dr. S. Pawletta, HS Wismar Dr. U. Berger, Leibniz-Institut für Atmosphärenphysik e.V., Kühlungsborn Dr. G. Baumgarten, Leibniz-Institut für Atmosphärenphysik e.V., Kühlungsborn

1. durch die Interpolation von Einzelbildern kann eine erneute Visualisierung von Daten vermieden werden;
2. durch indirekte Kombination der unterschiedlichen IDL-Grafiksysteme können ihre Vorteile in einer Visualisierung genutzt werden;
3. die erarbeitete Methode zur Umsetzung der Strömungsvisualisierung mit IDL kann auch auf andere Modellergebnisse des IAP angewendet werden;
4. komplexe Daten setzen komplexe Visualisierungsmethoden voraus, um enthaltene Informationen gut erfassbar zu machen;
5. eine ausführliche Beschreibung der Arbeitsumgebung erleichtert das Arbeiten in ihr;